

# **Induction of Fuzzy Rules from Chemical Process Data using Growing Neural Gas and Reactive Tabu Search Methods**

by

Francois Stefan Gouws

Dissertation presented for the Degree of  
Doctor of Philosophy in Engineering  
at the University of Stellenbosch

Promoter: Professor Chris Aldrich

October 1999



# Declaration

I the undersigned hereby declare that the work contained in this dissertation is my own original work and has not previously in its entirety or in part been submitted at any university for a degree.

Francois Stefan Gouws

31 October 1999



# Acknowledgements

My greatest professional debt is towards my supervisor Chris Aldrich and my good friend and colleague Gregor Schmitz. Chris taught me a great deal about the practicalities of research, gave me the freedom to pursue my ideas and sets a standard of language usage and report writing that I still aspire to.

Gregor assisted me in numerous aspects of my studies. The most important of these were teaching me to think more logically and critically about research, religion, philosophy and life in general, helping me to write the CORA software, allowing me to use his excellent linear regression code, critiquing my ideas and providing numerous useful ideas of his own.

I would like to thank past and current members of the Process Control Group at the University of Stellenbosch, JP Barnard, Etienne Hugo, Petrus Koller, Nelius Goosen and Dirko van Schalkwyk in particular, for helping to create a study environment which was fun to work and play in and answering my many questions regarding the meaning and truth of life on this planet, modelling, time series analysis, process control and Linux.

My appreciation goes towards the writers of the LEDA (<http://www.mpi-sb.mgp.de>) C++ library, parts of which are used by the Growing Neural Gas implementation that I wrote.

The Sasol Group of Companies supported me financially during both my undergraduate and postgraduate years as a student. In addition, I received a bursary from the Foundation of Research and Development during my postgraduate studies. I am extremely grateful towards both these institutions for their financial assistance.

My postgraduate years would have been much more dreary and uneventful without the arrival, enthusiasm, spontaneity and encouragement of my wife Santie. Santie, I love you and look forward to the rest of our lives together.

My parents gave me ceaseless support during my studies and created a home that was wonderful to stay and grow up in. My brother René fixed all the broken electronic equipment that I brought to him. My sister Estelle made music. Thanks to you all.

Finally, my greatest appreciation goes towards God. Lord, thank you for the privilege of being able to pursue a Ph.D. Thank you for all the gifts that You have given me and continue to shower upon me. Thank you for giving meaning and purpose to life. Thank you that You love me.

# Synopsis

The artificial intelligence community has developed a large body of algorithms that can be employed as powerful data analysis tools. However, such tools are not readily used in petrochemical plant operational decision support. This is primarily because the models generated by such tools are either too inaccurate or too difficult to understand if of acceptable accuracy. The Combinatorial Rule Assembler (CORA) algorithm is proposed to address these problems. The algorithm uses membership functions made by the Growing Neural Gas (GNG) radial basis function network training technique to assemble internally disjunctive, 0<sup>th</sup>-order Sugeno fuzzy rules using the nongreedy Reactive Tabu Search (RTS) combinatorial search method.

An evaluation of the influence of CORA training parameters revealed the following. First, for certain problems CORA models have an attribute space overlap that is one third of their GNG-generated counterparts. Second, the use of more fuzzy rules generally leads to better model accuracy. Third, decreased swap (or move) thresholds do not consistently lead to more accurate and / or simpler models. Fourth, utilisation of moves rather than swaps during rule antecedent assembly leads to better rule simplification. Fifth, consequent magnitude penalisation generally improves accuracy, especially if many rules are built. Variance of results is also usually reduced. Sixth, employing Yu rather than Zadeh operators leads to improved accuracy. Seventh, use of the GNG adjacency matrix significantly reduces the combinatorial complexity of rule construction. Eighth, AIC and BIC criteria used find the “right-sized” model exhibited local optima. Last, the CORA algorithm struggles to model problems that have a low exemplar to attribute ratio.

On a chaotic time series problem the CORA algorithm builds models that are significantly (with at least 95% confidence) more accurate than those generated using multiple linear regression (MLR), CART regression trees and multivariate adaptive regression splines (MARS). However, only the RTS component models are significantly more accurate than those of the GNG and  $k$ -means (RBF) radial basis function network methods. In terms of complexity, the CORA models were significantly simpler than the CART and RBF models but more complex than the MLR, MARS and multilayer perceptron models that were evaluated. Taking all results for this problem into account, it is the author’s opinion that the drop in accuracy (at worst 0.42%) of the CORA models, because of membership function merging and rule reduction, is justified by the increase in model simplicity (at least 22%). In addition, these results show that relatively intelligible “if...then...” fuzzy rule models can be built from chemical process data that are competitive (in terms of accuracy) with other, less intelligible, model types (e.g. multivariate spline models).

# Opsomming

'n Groot aantal algoritmes wat as kragtige data-analiseerders gebruik kan word, is tot op hede ontwikkel, veral deur navorsers op die gebied van skynintelligensie, waar 'n hoë premie geplaas word op die beskikbaarheid van doeltreffende soekmetodes. Die algoritmes word egter nie geredelik vir operasionele besluitnemingsondersteuning in petrochemiese aanlegte gebruik nie, omrede die modelle wat op sodanige wyse gegenereer is, óf te onakkuraat, óf indien akkuraat genoeg, te moeilik is om te verstaan. Die Combinatorial Rule Assembler (CORA) algoritme word in hierdie werk voorgehou as moontlike oplossing vir hierdie probleme. Die algoritme gebruik lidmaatskap-funksies wat m.b.v. 'n Groeiende Neurale Gas (GNG) algoritme opgestel is, om intern disjunktiewe, 0<sup>de</sup> orde Sugeno wasige logikareëls saam te stel deur gebruik te maak van 'n nie-gulsige kombinatoriese Reaktiewe Tabu Soektog (RTS) soek.

Die invloed van CORA leerparameters is ondersoek en daar is eerstens gevind dat CORA modelle vir sommige probleme intreeruite-oorvleuelings oplewer wat gelykstaande is aan sowat 'n derde van dié van GNG gegenereerde modelle. Tweedens, die gebruik van 'n groter aantal wasige reëls lei tot beter modusakkuraatheid. Derdens, verlaagde omruilperke lei nie altyd tot akkurater of eenvoudiger modelle nie. Vierdens, die gebruik van verplasings in plaas van omruilings tydens die samestelling van reëls lei tot beter reëlvereenvoudiging. Vyfdens, ordegrootte-penaliserings van die konsekwente van reëls dra oor die algemeen by tot beter akkuraatheid, veral wanneer baie reëls afgelei is. Die variansie van die resultate is ook verminder. Sedens, die implementering van Yu-, eerder as Zadeh-operators lei tot beter akkuraatheid. In die sewende plek, die gebruik van die GNG aangrensende matriks lei tot merkbaar laer kombinatoriese kompleksiteit in die aflei van wasige reëls. In die agste plek, die AIC en BIC kriteria, gebruik om die “regte grootte” model te vind, word in lokale optima vasgevang. Laastens vind die CORA algoritme dit moeilik om probleme te modelleer wanneer die verhouding van die aantal datapunte tot die aantal intreeveranderlikes laag is.

Die analise van 'n chaotiese tydreeksprobleem het aangetoon dat die CORA algoritme modelle bou wat beduidend (met ten minste 95% betroubaarheid) akkurater is as dié wat deur veelvoudige lineêre regressie (VLR), CART regressiebome en multiveranderlike aanpasbare regressielatfunksies (MARS) gebou is. Net die modelle van die RTS komponent van die CORA algoritme is egter beduidend meer akkuraat as die GNG en  $k$ -gemiddelde (RBF) radiale basis-funksiemodelle. In terme van kompleksiteit is die CORA modelle beduidend meer eenvoudig as

die CART en RBF modelle, maar meer kompleks as die VLR, MARS en multilaag-perseptronmodelle wat geëvalueer is. As alle resultate in ag geneem word, is dit die outeur se mening dat die vermindering in akkuraatheid (ten meeste 0.42%) van die CORA modelle, as gevolg van die samevoeging van lidmaatfunksies en die vermindering van reëls, geregverdig word deur die afname in die kompleksiteit van die model (ten minste 22%). Die resultate toon ook dat meer verstaanbare “as...dan...” wasige reël-modelle van chemiese prosess data gebou kan word as wat die geval is met bv. multiveranderlike latfunksiemodelle.

# Table of Contents

<b>Acknowledgements</b>	<b>i</b>
<b>Synopsis</b>	<b>ii</b>
<b>Opsomming</b>	<b>iii</b>
<b>Table of Contents</b>	<b>v</b>
<b>List of Figures</b>	<b>ix</b>
<b>List of Tables</b>	<b>xv</b>
<b>Nomenclature</b>	<b>xvi</b>
<b>1. Introduction</b>	<b>1</b>
1.1 Problem Statement.....	2
1.2 Aspects of Comprehensibility.....	4
1.3 Rule-based Description of Chemical Processes.....	5
1.4 The Combinatorial Rule Assembler Algorithm.....	5
1.5 The Modelling of a Chemical Process.....	6
1.6 Conclusions and Future Work.....	7
<b>2. Existing Rule Construction Techniques</b>	<b>8</b>
2.1 Describing Concepts with Rules.....	9
2.1.1 Crisp Rules.....	9
2.1.2 Fuzzy Rules.....	10
2.2 Crisp Rule Induction.....	15
2.2.1 Tree-based Rule Induction.....	15
2.2.2 Rule Induction by Covering.....	18
2.3 Fuzzy Rule Modelling.....	21
2.3.1 Evaluation of Publications Describing Fuzzy Methods.....	21
2.3.2 Criteria Used to Choose Fuzzy Rule Construction Methods.....	23
2.3.3 Fuzzy Rule Construction Using a Clustering Approach.....	23
2.3.4 The Induction of Fuzzy Decision Trees.....	26
2.3.5 Fuzzy Modelling using Genetic Algorithms.....	28
2.3.6 Neural Network Methods.....	32
2.4 Rule Extraction from Neural Networks.....	41
2.4.1 Crisp Rule Extraction Using TREPAN.....	42
2.4.2 The ANN-DT Rule Extraction Algorithm.....	44
2.5 Rule Construction Techniques and the Internet.....	45

<b>3.</b>	<b>Evaluation of Rule Construction Methods</b>	<b>46</b>
3.1	Limitations of Search Strategies that Build “if...then...” Rules .....	47
3.1.1	Computational Complexity of Rule Construction .....	48
3.1.2	Limitations of Rule Building by the Decision Tree Route .....	50
3.1.3	Limitations of Rule Building by the Set Covering Approach .....	54
3.2	Rule Comprehensibility .....	59
3.2.1	Historical Trends in Rule Format .....	60
3.2.2	The Changing Appearance of Crisp Decision Trees .....	61
3.2.3	The Changing Appearance of Fuzzy Rules .....	65
3.3	Discussion of Algorithmic Limitations.....	67
3.3.1	Analysis of Data with a Low Exemplar to Attribute Ratio.....	68
3.3.2	Results Obtained by Advanced Rule Construction Methods.....	68
3.3.3	Reasons for the Apparent Poor Performance of Advanced Rule Construction Techniques .....	72
3.3.4	Conclusions.....	73
3.3.5	A Final Note.....	75
<b>4.</b>	<b>The Combinatorial Rule Assembler</b>	<b>76</b>
4.1	The Growing Neural Gas Algorithm .....	77
4.1.1	The Unsupervised GNG Algorithm.....	80
4.1.2	The Supervised GNG Algorithm .....	84
4.2	The Reactive Tabu Search .....	87
4.2.1	Limit Cycle and Attractor Detection and Avoidance .....	87
4.2.2	Summary of RTS Scheme.....	88
4.2.3	The RTS Algorithm .....	89
4.2.4	The RTS Reactive Mechanisms.....	92
4.3	The CORA algorithm.....	94
4.3.1	Overview of the CORA Methodology.....	95
4.3.2	Reasons for Using the Growing Neural Gas Algorithm .....	99
4.3.3	Reasons for Using the Reactive Tabu Search Technique .....	101
4.3.4	Implementation Details of the CORA Algorithm.....	104
	<b>Evaluation of the CORA Technique</b>	<b>121</b>
5.1	Issues that are Investigated .....	122
5.1.1	Predictive Performance and Model Complexity.....	122
5.1.2	Influence of CORA Algorithm Training Parameters.....	124
5.2	Experimental Design.....	125
5.2.1	Current Methodology for Comparing Algorithms and Models .....	125
5.2.2	Experimental Method Used .....	126
5.2.3	Predictive Performance Estimation .....	127
5.3	Problems and Algorithms Evaluated.....	128
5.3.1	Data Sets Investigated.....	128

5.3.2	Algorithms Evaluated in Chapters 6 and 7 .....	133
5.3.3	Details of Experimental Procedure .....	135
<b>6.</b>	<b>Results and Discussion of the Empirical Evaluation of the CORA Algorithm</b>	<b>137</b>
6.1	CORA Fuzzy Rule Models vs. Models Derived by Other Algorithms .....	137
6.1.1	CORA Algorithm Training Capability .....	137
6.1.2	Fuzzy Rule Model Predictive Accuracy .....	141
6.1.3	Fuzzy Rule Model Complexity .....	148
6.2	Effect of CORA Algorithm Training Parameters .....	158
6.2.1	Number of Assembled Fuzzy Rules .....	159
6.2.2	Percentage of RTS Swaps or Moves Evaluated per Iteration .....	167
6.2.3	Rule Construction Using Swaps, Moves or Both .....	173
6.2.4	Consequent Magnitude Penalisation .....	179
6.2.5	Attribute Space Overlap Penalisation .....	184
6.2.6	Yu Fuzzy Operators vs. Zadeh Fuzzy Operators .....	188
6.3	Sundry Issues and a Discussion of Results .....	190
6.3.1	Minimisation of Attribute Space Overlap .....	190
6.3.2	RTS Component Training Characteristics .....	194
6.3.3	Selecting the Optimum GNG Radial Basis Function Network Model and Membership Function Merging .....	198
6.3.4	Variance of Results Generated by the CORA Algorithm .....	200
6.4	Summary and Conclusions .....	202
<b>7.</b>	<b>Modelling of a Chaotic Reaction System</b>	<b>207</b>
7.1	Purpose of Investigation .....	208
7.2	The Chaotic Reaction System .....	208
7.2.1	Description of the Reaction System .....	208
7.2.2	Data Generation .....	209
7.3	Experimental Design .....	210
7.3.1	Experimental Method .....	210
7.3.2	Models and Modelling Techniques Evaluated .....	211
7.3.3	Hypothesis Testing and Statistical Measures .....	212
7.4	Results .....	213
7.4.1	Comparative Predictive Accuracy .....	213
7.4.2	Comparative Model Complexity .....	216
7.4.3	Interpretation of a CORA Fuzzy Rule .....	219
7.4.4	Fuzzy Rule Model Time Series Prediction .....	221
7.5	Summary and Conclusions .....	223
<b>8.</b>	<b>Conclusions and Future Work</b>	<b>225</b>
8.1	Evaluation of Existing Rule Building Techniques .....	225
8.2	Evaluation of the CORA Algorithm .....	227
8.3	Future Work .....	232



<b>References</b>	<b>234</b>
<b>A Default Algorithm Parameter Settings</b>	<b>251</b>
A.1 The CART Decision Tree Algorithm.....	251
A.2 Crisp Rule Induction Using BEXA.....	252
A.3 The Growing Neural Gas (GNG) Algorithm .....	253
A.4 The Combinatorial Rule Assembler (CORA).....	253
A.5 The k-means Radial Basis Function Network.....	255
A.6 The Multilayer Perceptron .....	256
A.7 Multivariate Adaptive Regression Splines.....	256
<b>B Additional Figures for the Empirical Evaluation Presented in Chapter 6</b>	<b>257</b>
<b>C Algorithmic Training Parameters Used in the Evaluation of the Autocatalytic Data</b>	<b>289</b>
C.1 The CART Algorithm .....	289
C.2 The Growing Neural Gas Algorithm .....	290
C.3 The CORA Algorithm.....	290
C.4 The k-means Radial Basis Function Network.....	290
C.5 The Multilayer Perceptron .....	291
C.6 Multivariate Adaptive Regression Splines.....	291



# List of Figures

2.1	Hyperspherical and Hyperellipsoidal two-dimensional Gaussians .....	13
2.2	A Typical Classification Tree .....	15
2.3	Graphical Representation of a Neural Network.....	32
2.4	Fuzzy ARTMAP Structure .....	40
4.1	Two Ways of Defining Proximity among a Set of Data Points.....	79
4.2	Initialisation Function of the Unsupervised GNG Algorithm .....	80
4.3	Main Unsupervised GNG Procedure .....	81
4.4	Unsupervised GNG Cell Node Insertion Function .....	82
4.5	Unsupervised Learning of an Attribute Manifold.....	83
4.6	Initialisation Function of Supervised GNG .....	85
4.7	Main Procedure of the Supervised GNG Algorithm .....	86
4.8	Cell Node Insertion Function of the GNG Supervised Algorithm .....	87
4.9	RTS Initialisation Function.....	90
4.10	Main RTS Procedure .....	91
4.11	The RTS <i>check_for_repetitions</i> Function.....	92
4.12	The <i>best_move</i> Function of the RTS Technique .....	93
4.13	The RTS <i>diversify_search</i> Function.....	94
4.14	Zadeh Disjunction Operator vs. Yu Disjunction Operator .....	97
4.15	Zadeh Conjunction Operator vs. Yu Conjunction Operator .....	98
4.16	Membership Function Merging using GNG generated Radial Basis Functions .....	99
4.17	Adapted <i>insert_cell_node</i> Function of the GNG algorithm .....	105
4.18	Merging Two Membership Functions that Intersect Once .....	110
4.19	Merging Two Membership Functions that Intersect Twice.....	111
6.1	Estimated Output of the GNG Fuzzy Rule Model.....	140
6.2	Estimated Output of the CORA Fuzzy Rule Model .....	140
6.3	Output Prediction Surface of a Typical Fuzzy Rule Generated by the CORA Algorithm for the Spiral Problem .....	141
6.4	Predictive Accuracy on the Abalone Problem.....	143
6.5	Predictive Accuracy on the Auto Problem .....	143
6.6	Predictive Accuracy on the Housing Problem.....	143
6.7	Predictive Accuracy on the Servo Problem .....	143
6.8	Predictive Accuracy on the Sincos Problem.....	144
6.9	Predictive Accuracy on the Ionosphere Problem.....	144
6.10	Predictive Accuracy on the Slugflow Problem.....	144
6.11	Predictive Accuracy on the Pima Problem .....	144
6.12	Predictive Accuracy on the WBC Problem .....	145

6.13	Number of Rules on Abalone and Auto Problems .....	151
6.14	Number of Rules on Housing and Servo Problems .....	151
6.15	Number of Rules on Ionosphere and Slugflow Data .....	151
6.16	Number of Rules on Pima and WBC Problems.....	151
6.17	Number of Concepts on Abalone and Auto Problems.....	152
6.18	Number of Concepts on Housing and Servo Problems .....	152
6.19	Number of Concepts on Ionosphere and Slugflow Data .....	152
6.20	Number of Concepts on Pima and WBC Problems.....	152
6.21	Number of Parameters on Abalone and Auto Problems.....	153
6.22	Number of Parameters on Housing and Servo Data .....	153
6.23	No. of Parameters on Ionosphere and Slugflow Data.....	153
6.24	Number of Parameters on Pima and WBC Problems .....	153
6.25	Input Space Overlap on Abalone and Auto Problems .....	154
6.26	Input Space Overlap on Housing and Servo Problems.....	154
6.27	Input Space Overlap on Ionosphere and Slugflow Data.....	154
6.28	Input Space Overlap on Pima and WBC Problems .....	154
6.29	Accuracy vs. Number of Rules for Abalone Problem .....	160
6.30	Accuracy vs. Number of Rules for Auto Problem.....	160
6.31	No. of Concepts vs. No. of Rules for Abalone Problem.....	160
6.32	No. of Concepts vs. No. of Rules for Auto Problem .....	160
6.33	Accuracy vs. Number of Rules for Housing Problem .....	161
6.34	Accuracy vs. Number of Rules for Servo Problem .....	161
6.35	No. of Concepts vs. No. of Rules for Housing Problem.....	161
6.36	No. of Concepts vs. No. of Rules for Servo Problem.....	161
6.37	Accuracy vs. Number of Rules for Ionosphere Problem .....	162
6.38	Accuracy vs. Number of Rules for Pima Problem.....	162
6.39	No. of Concepts vs. No. of Rules for Ionosphere Data .....	162
6.40	No. of Concepts vs. No. of Rules for Pima Problem .....	162
6.41	Accuracy vs. Search Resolution for Auto Data .....	169
6.42	Accuracy vs. Search Resolution for Servo Data.....	169
6.43	No. of Concepts vs. Search Resolution for Auto Data .....	169
6.44	No. of Concepts vs. Search Resolution for Servo Data.....	169
6.45	Accuracy vs. Search Resolution for Pima Data.....	170
6.46	Accuracy vs. Search Resolution for Slugflow Data .....	170
6.47	No. of Concepts vs. Search Resolution for Pima Data .....	170
6.48	No. of Concepts vs. Search Resolution for Slugflow Data.....	170
6.49	Training Accuracy and RTS Search Resolution against RTS Iterations .....	172
6.50	Accuracy vs. SMB for Abalone Problem (30 rules).....	174
6.51	Accuracy vs. SMB for Auto Problem (27 rules) .....	174
6.52	No. of Concepts vs. SMB for Abalone Data (30 rules) .....	174
6.53	No. of Concepts vs. SMB for Auto Problem (27 rules).....	174

6.54	Accuracy vs. SMB for Housing Problem (30 rules).....	175
6.55	Accuracy vs. SMB for Servo Problem (15 rules) .....	175
6.56	Concepts vs. SMB for Housing Problem (30 rules) .....	175
6.57	No. of Concepts vs. SMB for Servo Data (15 rules) .....	175
6.58	Accuracy vs. SMB for Ionosphere Problem (30 rules).....	176
6.59	Accuracy vs. SMB for Pima Problem (15 rules) .....	176
6.60	Concepts vs. SMB for Ionosphere Problem (30 rules) .....	176
6.61	Concepts vs. SMB for Pima Problem (15 rules).....	176
6.62	Accuracy vs. SMB for Slugflow Problem (6 rules).....	177
6.63	No. of Concepts vs. SMB for Slugflow Data (6 rules) .....	177
6.64	Accuracy vs. Use of Weight Penalty for Abalone Data.....	181
6.65	Accuracy vs. Use of Weight Penalty for Auto Problem .....	181
6.66	Concepts vs. Use of Weight Penalty for Abalone Data.....	181
6.67	Concepts vs. Use of Weight Penalty for Auto Problem .....	181
6.68	Accuracy vs. Use of Weight Penalty for Housing Data .....	182
6.69	Accuracy vs. Use of Weight Penalty for Servo Problem.....	182
6.70	Concepts vs. Use of Weight Penalty for Housing Data.....	182
6.71	Concepts vs. Use of Weight Penalty for Servo Problem .....	182
6.72	Accuracy vs. Use of Weight Penalty for Slugflow Data .....	183
6.73	Accuracy vs. Use of Weight Penalty for Pima Problem.....	183
6.74	Concepts vs. Use of Weight Penalty for Slugflow Data.....	183
6.75	Concepts vs. Use of Weight Penalty for Pima Problem .....	183
6.76	Accuracy vs. Use of Overlap Penalty (Abalone Data) .....	186
6.77	Accuracy vs. Use of Overlap Penalty (Auto Data).....	186
6.78	Concepts vs. Use of Overlap Penalty (Abalone Data).....	186
6.79	Concepts vs. Use of Overlap Penalty (Auto Data) .....	186
6.80	Accuracy vs. Attribute Space Overlap for Pima Problem .....	187
6.81	Number of Concepts vs. Attribute Space Overlap for Pima Problem .....	187
6.82	Accuracy vs. Use of Yu or Zadeh Fuzzy Operators .....	189
6.83	Concepts vs. Use of Yu or Zadeh Fuzzy Operators.....	189
6.84	Parameters vs. Use of Yu or Zadeh Fuzzy Operators.....	189
6.85	Accuracy vs. Overlap Level (Abalone and Auto Data) .....	192
6.86	Accuracy vs. Overlap Level (Housing and Servo Data).....	192
6.87	Concepts vs. Overlap Level (Abalone and Auto Data).....	192
6.88	Concepts vs. Overlap Level (Housing and Servo Data) .....	192
6.89	GNG-generated Adjacency Matrix for Auto Problem Using Thirty Rules .....	194
6.90	Fitness, Training $R^2$ and Search Resolution for the Abalone Problem.....	196
6.91	Fitness, Training $R^2$ and Search Resolution for the Auto Problem .....	196
6.92	Fitness, Training $R^2$ and Search Resolution for the Housing Problem.....	197
6.93	Fitness, Training $R^2$ and Search Resolution for the Servo Problem.....	197
6.94	Validation Accuracy and Information Criteria Results vs. Number of GNG Fuzzy Rules .....	199

6.95	Variance of Predictive Accuracy Results vs. Data Set Density.....	201
7.1	Predictive Accuracy of CORA Algorithm and Other Modelling Techniques.....	213
7.2	CORA (Minimum Overlap Variant) Fuzzy Rule Assembled by the RTS Component.....	219
7.3	CORA (Minimum Overlap Variant) Fuzzy Rule Obtained after Membership Function Merging .....	220
7.4	Estimated vs. Target Autocatalytic Times Series .....	222
B.1	No. of Parameters vs. No. of Rules for Abalone Data.....	260
B.2	No. of Parameters vs. No. of Rules for Auto Data .....	260
B.3	No. of Parameters vs. No. of Rules for Housing Data.....	260
B.4	No. of Parameters vs. No. of Rules for Servo Problem .....	260
B.5	No. of Parameters vs. No. of Rules for Ionosphere Data.....	261
B.6	No. of Parameters vs. No. of Rules for Pima Problem .....	261
B.7	Accuracy vs. No. of Rules for Auto Problem (0.5) .....	262
B.8	Accuracy vs. No. of Rules for Servo Problem (0.5) .....	262
B.9	Accuracy vs. No. of Rules for Auto Problem (0.01) .....	262
B.10	Accuracy vs. No. of Rules for Servo Problem (0.01) .....	262
B.11	No. of Concepts vs. No. of Rules for Auto Data (0.5).....	263
B.12	No. of Concepts vs. No. of Rules for Servo Data (0.5) .....	263
B.13	No. of Concepts vs. No. of Rules for Auto Data (0.01).....	263
B.14	No. of Concepts vs. No. of Rules for Servo Data (0.01) .....	263
B.15	No. of Parameters vs. No. of Rules for Auto Data (0.5).....	264
B.16	No. of Parameters vs. No. of Rules for Servo Data (0.5) .....	264
B.17	No. of Parameters vs. No. of Rules for Auto Data (0.01).....	264
B.18	Parameters vs. No. of Rules for Servo Data (0.01).....	264
B.19	Accuracy vs. No. of Rules for Pima Problem (0.5) .....	265
B.20	No. of Concepts vs. No. of Rules for Pima Data (0.5) .....	265
B.21	Accuracy vs. No. of Rules for Pima Problem (0.01) .....	265
B.22	No. of Concepts vs. No. of Rules for Pima Data (0.01) .....	265
B.23	No. of Parameters vs. No. of Rules for Pima Problem (0.5) .....	266
B.24	No. of Parameters vs. No. of Rules for Pima Problem (0.01) .....	266
B.25	No. of Parameters vs. Search Resolution (Auto (27)) .....	267
B.26	No. of Parameters vs. Search Resolution (Servo (15)).....	267
B.27	No. of Parameters vs. Search Resolution (Pima (15)).....	267
B.28	Parameters vs. Search Resolution (Slugflow (6)).....	267
B.29	Accuracy vs. Search Resolution (Auto (22)) .....	268
B.30	Accuracy vs. Search Resolution (Servo (10)).....	268
B.31	Accuracy vs. Search Resolution (Auto (17)).....	268
B.32	Accuracy vs. Search Resolution (Servo (5)).....	268
B.33	No. of Concepts vs. Search Resolution (Auto (22)) .....	269

B.34 No. of Concepts vs. Search Resolution (Servo (10))	269
B.35 No. of Concepts vs. Search Resolution (Auto (17))	269
B.36 No. of Concepts vs. Search Resolution (Servo (5))	269
B.37 No. of Parameters vs. Search Resolution (Auto (22))	270
B.38 No. of Parameters vs. Search Resolution (Servo (10))	270
B.39 No. of Parameters vs. Search Resolution (Auto (17))	270
B.40 No. of Parameters vs. Search Resolution (Servo (5))	270
B.41 Accuracy vs. Search Resolution (Pima (10))	271
B.42 No. of Concepts vs. Search Resolution (Pima (10))	271
B.43 Accuracy vs. Search Resolution (Pima (5))	271
B.44 No. of Concepts vs. Search Resolution (Pima (5))	271
B.45 No. of Parameters vs. Search Resolution (Pima (10))	272
B.46 No. of Parameters vs. Search Resolution (Pima (5))	272
B.47 No. of Parameters vs. SMB for Abalone Data (30)	273
B.48 No. of Parameters vs. SMB for Auto Data (27)	273
B.49 No. of Parameters vs. SMB for Housing Data (30)	273
B.50 Parameters vs. SMB for Servo Problem (15)	273
B.51 No. of Parameters vs. SMB for Ionosphere Data (30)	274
B.52 No. of Parameters vs. SMB for Pima Data (15)	274
B.53 No. of Parameters vs. SMB for Slugflow Data (6)	275
B.54 Accuracy vs. SMB for Abalone Problem (20)	276
B.55 Accuracy vs. SMB for Auto Problem (22)	276
B.56 Accuracy vs. SMB for Abalone Problem (10)	276
B.57 Accuracy vs. SMB for Auto Problem (17)	276
B.58 No. of Concepts vs. SMB for Abalone Problem (20)	277
B.59 No. of Concepts vs. SMB for Auto Problem (22)	277
B.60 No. of Concepts vs. SMB for Abalone Problem (10)	277
B.61 No. of Concepts vs. SMB for Auto Problem (17)	277
B.62 No. of Parameters vs. SMB for Abalone Problem (20)	278
B.63 No. of Parameters vs. SMB for Auto Problem (22)	278
B.64 No. of Parameters vs. SMB for Abalone Problem (10)	278
B.65 No. of Parameters vs. SMB for Auto Problem (17)	278
B.66 Accuracy vs. SMB for Housing Problem (20)	279
B.67 Accuracy vs. SMB for Servo Problem (10)	279
B.68 Accuracy vs. SMB for Housing Problem (10)	279
B.69 Accuracy vs. SMB for Servo Problem (5)	279
B.70 No. of Concepts vs. SMB for Housing Problem (20)	280
B.71 No. of Concepts vs. SMB for Servo Problem (10)	280
B.72 No. of Concepts vs. SMB for Housing Problem (10)	280
B.73 No. of Concepts vs. SMB for Servo Problem (5)	280
B.74 No. of Parameters vs. SMB for Housing Problem (20)	281



B.75 No. of Parameters vs. SMB for Servo Problem (10) .....	281
B.76 No. of Parameters vs. SMB for Housing Problem (10) .....	281
B.77 No. of Parameters vs. SMB for Servo Problem (5) .....	281
B.78 Accuracy vs. SMB for Ionosphere Problem (20) .....	282
B.79 Accuracy vs. SMB for Pima Problem (10) .....	282
B.80 Accuracy vs. SMB for Ionosphere Problem (10) .....	282
B.81 Accuracy vs. SMB for Pima Problem (5) .....	282
B.82 No. of Concepts vs. SMB for Ionosphere Data (20) .....	283
B.83 No. of Concepts vs. SMB for Pima Problem (10) .....	283
B.84 No. of Concepts vs. SMB for Ionosphere Data (10) .....	283
B.85 No. of Concepts vs. SMB for Pima Problem (5) .....	283
B.86 No. of Parameters vs. SMB for Ionosphere Data (20) .....	284
B.87 No. of Parameters vs. SMB for Pima Problem (10) .....	284
B.88 No. of Parameters vs. SMB for Ionosphere Data (10) .....	284
B.89 No. of Parameters vs. SMB for Pima Problem (5) .....	284
B.90 Parameters vs. Weight Penalty Use for Abalone Data .....	285
B.91 Parameters vs. Weight Penalty Use for Auto Problem .....	285
B.92 Parameters vs. Weight Penalty Use for Housing Data .....	285
B.93 Parameters vs. Weight Penalty Use for Servo Problem .....	285
B.94 Parameters vs. Use of Weight Penalty for Slugflow Problem .....	286
B.95 Parameters vs. Use of Weight Penalty for Pima Problem .....	286
B.96 Parameters vs. Use of Overlap Penalty (Abalone Data) .....	287
B.97 Parameters vs. Use of Overlap Penalty (Auto Data) .....	287
B.98 Parameters vs. Use of Overlap Penalty (Pima Data) .....	287
B.99 Parameters vs. Overlap Level (Abalone and Auto Problems) .....	288
B.100 Parameters vs. Overlap Level (Housing and Servo Problems) .....	288

# List of Tables

3.1	Average Improvement in Predictive Accuracy of Advanced Decision Tree Algorithms over Univariate Decision Tree Algorithms .....	64
3.2	Predictive Classification Accuracy and Model Complexity Results .....	70
3.3	Predictive Performance Using Different Levels of Lookahead.....	71
4.1	Performance of Combinatorial Search Methods on $N$ - $K$ Problems.....	102
4.2	Performance on Small and Large Multiknapsack Problems.....	103
4.3	Algorithm Performance on Strongly Correlated Tasks .....	104
5.1	Details of the Data Sets Studied in Chapter 6.....	128
6.1	Parameter Settings for Number of Assembled Fuzzy Rules Experiments .....	163
6.2	Average Decrease in Predictive Accuracy and Model Complexity Owing to Membership Function Merging .....	165
6.3	Average Model Complexity Decrease as a Result of Rule Reduction .....	167
6.4	Comparison of GNG and CORA Algorithm Attribute Space Overlap .....	191
7.1	Predictive Accuracy Significance Results .....	214
7.2	Statistical Significance Critical Confidence Levels for the Difference in Property Values of the CORA Models .....	216
7.3	Comparative Model Complexity in Terms of Number of “if...then...” Rules, Number of Concepts and Attribute Space Overlap .....	216
7.4	Comparative Model Parameter Complexity .....	216
7.5	Significance of Comparative Model Complexity Results .....	217
7.6	Average Percentage Decrease (Increase) in CORA Model Property Value vs. GNG Models .....	218
A.1	BEXA Parameter Values and Settings.....	252
A.2	GNG Parameter Values and Settings.....	253
A.3	CORA Algorithm Parameter Values .....	254
A.4	$k$ -means Radial Basis Function Network Settings.....	255
A.5	Multilayer Perceptron Training Parameter Settings .....	256
A.6	MARS Settings .....	256
C.1	CORA Algorithm Training Parameter Values Used in Chapter 7.....	290

# Nomenclature

The nomenclature information presented here is divided according to the chapters of this dissertation. There is no nomenclature information for chapter 1. The numerical reference after each symbol description indicates the section in the dissertation where a particular symbol is first utilised. In a few cases, the same symbol is used in more than one context. In such cases the description of the symbol given here as well as where the symbol is used, should be used to determine which symbol interpretation is applicable. Symbols in bold typically represent multivalued variables such as vectors or matrices.

## Chapter 2 Nomenclature

$A$	Positive matrix that represents the metric of a multidimensional Gaussian	2.1.2.2
$A_C$	Subset of consistent antecedents of $A$ where $A$ is the set of all $VL_1$ antecedents that exist for a given problem	2.2.2.2
$A_j$	Data attribute	2.1.1
$A_M$	Set of most general consistent antecedents	2.2.2.2
$a$	Membership function parameter	2.1.2.2
$a_j$	Attribute value $j$ of attribute $A_j$	2.1.1
$B$	Arbitrary set	2.2.2.2
$B_j$	Data attribute	2.1.1
$b$	Membership function parameter	2.1.2.2
$b_j$	Attribute value $j$ of attribute $B_j$	2.1.1
$C$	Set cover	2.2.2.2
$C_j$	Data attribute	2.1.1
$c$	Membership function parameter	2.1.2.2
$c$	Arbitrary antecedent	2.2.2.2
$c$	Centre coordinate vector of a multidimensional Gaussian	2.1.2.2
$c_i$	Centre of the lefthand or righthand shoulder of a two-sided Gaussian	2.1.2.2
$c_j$	Attribute value $j$ of attribute $C_j$	2.1.1
$c_{ij}$	Centre coordinates of cluster $ij$	2.3.3.1
$d$	Membership function parameter	2.1.2.2
$E$	Data attribute	2.2.2.2



$e_j$	Attribute value $j$ of attribute $E$	2.2.2.2
$f$	Linear function of linguistic variables	2.1.2.3
$F_{0 \text{ or } 1}$	ART network layers	2.3.6.6
$G_i$	Problem attribute $i$	2.3.6.1
$H$	Problem output	2.3.6.1
$g$	Syntactic rule for generating linguistic terms	2.1.2.1
$I$	Complement coding of a data exemplar	2.3.6.6
$M$	Metric of multidimensional Gaussian	2.1.2.2
$m$	Semantic rule that associates a fuzzy set $m(t)$ with a linguistic term $t$	2.1.2.1
$N$	Set of exemplars with a negative output class	2.2.2.2
$N_{min}$	User-defined minimum number of data exemplars	2.3.3.1
$N_{max}$	User-defined maximum number of data exemplars	2.3.3.1
$n$	Number of logical tests in an $p$ -of- $n$ Boolean expression	2.1.1
$p$	Integer threshold of an $p$ -of- $n$ Boolean expression	2.1.1
$Q_{ij}$	Covariance matrix for cluster $ij$	2.3.3.1
$R$	Subset of attribute values excluded from $mga$ to obtain $c$ where $c$ is an arbitrary antecedent. $mga$ is the most general antecedent that covers a given set of data exemplars	2.2.2.2
$r(w_j, I)$	ART network match function for weight vector $w_j$ and complement coding $I$ and of category node $j$	2.3.6.6
$T$	Set of linguistic terms of a linguistic variable	2.1.2.1
$T_j$	ART network choice function for category node $j$	2.3.6.6
$t_i$	Linguistic term name	2.1.2.3
$u$	Arbitrary linguistic variable	2.3.6.4
$v$	Linguistic variable name	2.1.2.1
$w_j$	Vector of adaptive weights for category node $j$	2.3.6.6
$w$	Arbitrary linguistic variable	2.3.6.4
$X$	Universal set	2.1.2.1
$x$	Single member of universal set $X$	2.1.2.1
$\mu_m$	Membership function of a linguistic term	2.1.2.1
$\rho$	ART network vigilance criterion	2.3.6.6
$\sigma$	Width of a Gaussian membership function	2.1.2.2
$\sigma_i$	Width of the lefthand or righthand shoulder of a two-sided Gaussian	2.1.2.2

## Chapter 3 Nomenclature

$A$	Arbitrary concept	3.1.2.2
$B$	Arbitrary concept	3.1.2.2
$C$	Arbitrary concept	3.1.2.2
$D$	Arbitrary concept	3.1.2.2
$f$	Arbitrary Gaussian membership function of a combinations of original problem attributes	3.2.3
$g$	Arbitrary Gaussian membership function of a combinations of original problem attributes	3.2.3
$h$	Linear function of original problem attributes	3.2.3
$k$	Number of independent variables	3.1.1

## Chapter 4 Nomenclature

$A$	Adjacency matrix	4.1
$A$	Fuzzy rule model firing strength matrix	4.3.4.3
$A$	Set of lateral connections of network graph	4.1
$A^{(t)}$	Set of admissible moves at RTS <sup>1</sup> search iteration $t$ that may be used to generate new solutions	4.2.2
$a_i$	Age variable of neighbourhood connection $i$	4.1.1
$a_{max}$	User-defined maximum allowed age of neighbourhood connection	4.1.1
$b$	Vector representing the output of a set of data exemplars	4.3.4.3
$b'$	Vector representing the estimated (calculated) output of a set of data exemplars	4.3.4.3
$C$	Set of often-repeated moves	4.2.3
$C$	Augmented matrix used for linear regression	4.3.4.3
$c_j$	Reference vector of GNG <sup>2</sup> cell node $j$	4.1

---

<sup>1</sup> The RTS acronym stands for “Reactive Tabu Search”.

<sup>2</sup> The GNG acronym stands for “Growing Neural Gas”.

$d$	Number of problem attributes	4.3.4.8
$E$	Cost function for RTS algorithm	4.2.2
$E_b$	Cost function value for the best solution discovered thusfar by the RTS search, i.e. $f_b$	4.2.3
$F$	Set of feasible solutions for RTS algorithm	4.2.2
$f$	Arbitrary solution found using RTS	4.2.2
$f_b$	Best solution discovered by the RTS search thusfar	4.2.3
$f^{(t)}$	Solution found by the RTS at iteration $t$	4.2.2
$G$	Network graph	4.1
$g$	Arbitrary solution found using RTS	4.2.2
$g$	Number of problem attributes times the number of Gaussians in the hidden layer of the trained GNG network	4.3.4.8
$L$	Length of binary string	4.2.2
$i$	Vertex $i$ of network graph $G$	4.1
$M$	Attribute manifold	4.1
$M$	Set of elementary RTS moves	4.2.2
$m$	Number of data exemplars	4.3.4.3
$N_j$	Set of direct topological neighbours of cell node $j$	4.1.1
$N_w$	Set of direct topological neighbours of winner cell node $w(\mathbf{x})$	4.1.1
$N(f)$	Neighbourhood of a solution $f$	4.2.2
$n$	Arbitrary variable	4.1
$n$	Number of problem attributes	4.3.4.3
$n$	Number of fuzzy rules	4.3.4.3
$n_x$	Number of training exemplars	4.1.1
$P$	Product matrix used for linear regression	4.3.4.3
$p$	Number of model parameters	4.3.4.4
$\mathcal{R}^d$	$d$ -dimensional attribute space	4.1
$q$	Cell node with globally maximum resource	4.1.1
$R$	Current limit cycle length	4.2.2
$R_{avg}$	Moving average of the repetition interval (RTS algorithm)	4.2.3
$r_j$	Resource variable of cell node $j$	4.1.1
$r_w$	Resource variable of winner cell node $w(\mathbf{x})$	4.1.1
$S$	Set of synaptic weight vectors	4.1

$S^{(t)}$	Set of prohibited moves at RTS search iteration $t$	4.2.3
$s(x)$	Cell node with reference vector second closest to data exemplar $x$	4.1.1
$T^{(t)}$	Set of prohibited moves for RTS search iteration $t$	4.2.2
$t$	Maximum resource cell node that is a direct topological neighbour of $q$	4.1.1
$t_s$	Iteration when $S^{(t)}$ (the set of prohibited moves) was last modified	4.2.3
$V_j$	Voronoi polyhedron belonging to reference vector $c_j$	4.1
$V_j^{(M)}$	Masked Voronoi polyhedron of Voronoi polyhedron $V_j$	4.1
$u$	New cell node	4.1.1
$v$	Element of manifold $M$	4.1
$w(x)$	Cell node with reference vector closest to data exemplar $x$	4.1.1
$x$	Arbitrary data exemplar	4.1
$\alpha$	Cell node resource update factor for new cell node addition	4.1.1
$\beta$	Cell node resource update factor	4.1.1
$\varepsilon_w$	Reference vector update factor of winner cell node $w(x)$	4.1.1
$\varepsilon_n$	Reference vector update factor of winner cell node's direct topological neighbours	4.1.1
$\eta$	Multiple of training exemplars presented to the GNG algorithm before training is switched from unsupervised to supervised training	4.3.4.1
$\lambda$	Number of training exemplars presented to the GNG network before addition of new cell node	4.1.1
$\mu$	Elementary move applied a RTS solution to generate a new solution	4.2.2
$\mu^{(t)}$	Best (optimal cost function value) elementary move discovered during RTS search iteration $t$	4.2.2
$\sigma_i$	Width of multidimensional Gaussian $i$	4.1.1
$\tau$	Fraction of the mean length of all connections emanating from a specific cell node	4.1.2
$\Delta(i,j,k)$	Triangle with vertices $i, j$ and $k$	4.1
$\Phi(f)$	Number of times the solution $f$ has been visited in the current search trajectory	4.2.3
$\Lambda(\mu)$	Last RTS iteration when $\mu$ (an elementary move) was used	4.2.3
$\Pi(f)$	Last RTS iteration when the solution $f$ was encountered	4.2.3

## Chapter 5 Nomenclature

$a$	Parameter of Sincos function	5.3.1.7
$b$	Parameter of Sincos function	5.3.1.7
$c$	Parameter of Sincos function	5.3.1.7
$n$	Number of data exemplars	5.2.3
$R^2$	Root mean square error	5.2.3
$s$	Continuous attribute of Sincos function	5.3.1.7
$x$	Continuous attribute of Sincos function	5.3.1.7
$y_i$	True output of data exemplar $i$	5.2.3
$\hat{y}_i$	Estimated (calculated) output of data exemplar $i$	5.2.3
$\varepsilon$	Continuous attribute of Sincos function	5.3.1.7
$\phi$	Discrete attribute of Sincos function	5.3.1.7
$\theta$	Continuous attribute of Sincos function	5.3.1.7

## Chapter 6 Nomenclature

$B_j$	Data attribute	6.1.3
$b_j$	Attribute value $j$ of attribute $B_j$	6.1.3
$n$	Number of model parameters	6.1.3
$\tau$	Fraction of the mean length of all connections emanating from a specific cell node	6.1.3

# Chapter 7 Nomenclature

$A$	Chemical species	7.2.1
$a$	Operand of fuzzy AND operation	7.4.3
$B$	Chemical species	7.2.1
$b$	Operand of fuzzy AND operation	7.4.3
$C$	Chemical species	7.2.1
$C_j$	Concentration of chemical species $j$	7.2.1
$C_{j0}$	Reactor inlet concentration of chemical species $j$	7.2.1
$D$	Chemical species	7.2.1
$Da_j$	Damköhler number for chemical species $j$	7.2.1
$k$	Number of experiments performed for t-test analysis	7.3.3
$k_j$	Reaction rate constant for the reaction rate equation for chemical species $j$	7.2.1
$n_i$	Number of experiments performed by algorithm $i$ for t-test analysis	7.3.3
$Q$	Reactor feed flow rate	7.2.1
$r$	Problem degrees of freedom	7.3.3
$r_j$	Reaction rate of chemical species $j$	7.2.1
$t$	Time	7.2.1
$t_2$	Paired-sample t-test statistic	7.3.3
$V$	Reactor volume	7.2.1
$X$	Dimensionless concentration of chemical species A	7.2.1
$\bar{x}_i$	Mean of a set of model property values for algorithm $i$	7.3.3
$Y$	Dimensionless concentration of chemical species B	7.2.1
$Z$	Dimensionless concentration of chemical species C	7.2.1
$\alpha$	t-test confidence level	7.3.3
$\alpha_j$	Ratio of the concentration of two chemical species	7.2.1
$\sigma_i^2$	Variance of a set of model property values for algorithm $i$	7.3.3
$\tau$	Dimensionless time	7.2.1

# Chapter 1

## Introduction

Over the last decade a large number of techniques have been developed by the artificial intelligence community that are able to build models relating independent variables, or attributes, of some chemical or metallurgical system to a dependent variable, or output, of the system. These techniques are typically used in the analysis of complex processes for which there do not exist adequate fundamental models.

For such processes there are two primary repositories of knowledge regarding actual process operation. The first is the heuristic knowledge of process experts and human operators of the chemical process. Unfortunately, traditional knowledge acquisition from human experts and operators has proved problematic. It is often difficult to extract heuristic knowledge from process experts since they do not always base their decisions on precise facts or rules. They therefore find it difficult to express their knowledge in terms of concise, “if...then...” type rules (Muggleton, 1990; Fayyad and Irani, 1992). Furthermore, it has been found that different experts exhibit cognitive biases such as overconfidence and oversimplification (Johannsen and Alty, 1991; Kattan, 1994). Moreover, if the chemical process domain necessitates reasoning under uncertainty, viz. probabilistic reasoning, humans are known to be inconsistent in their description of subjective probabilities (Kahneman, et al., 1982). In addition, the knowledge human experts have is often incomplete and episodic rather than systematic (Sun, 1994). This means that it may take considerable time and involve significant expense to build an accurate and comprehensive knowledge base of such a complex, chemical process (Donald, 1994).

The second repository of process knowledge is the vast amount of measured data that are recorded in most modern chemical and metallurgical plants. Such data are usually stored

electronically and are therefore easily accessible. Furthermore, computer processing power continues to become increasingly inexpensive. For these reasons, and owing to the problems associated with traditional human knowledge acquisition, data analysis techniques, including those developed by the artificial intelligence community, are becoming increasingly popular as tools to obtain knowledge about complex chemical processes.

Applications of artificial intelligence modelling techniques are numerous and include the use of neural networks for metal corrosion prediction (Smets and Bogaerts, 1995), flotation froth interpretation (Moolman, et al., 1995), gas loop modelling and optimisation (Joubert, et al., 1996) and for chemical reactor modelling (Vaidyanathan and Venkatasubramanian, 1992; Shaw, et al., 1997). Decision tree techniques have also been applied to chemical process modelling. Examples are the automation of materials scheduling in a steel mill (Michie, 1992) and the improvement of process performance (Saraiva and Stephanopoulos, 1992) and productivity (Evans and Fisher, 1994). Rule-based techniques have also been used in the modelling of a number of chemical systems. These include the derivation of crisp rules to assist in material corrosion behaviour analysis (Koch and Fehsenfeld, 1995). Further examples are the construction of fuzzy rules that model metallurgical equipment (Karr and Weck, 1996) and that provide operational decision support (Wang, et al., 1997).

## 1.1 Problem Statement

Based on the above information one would expect such techniques to be quickly implemented as data analysis tools and in particular for operational decision support in the highly competitive chemical and metallurgical industries. Discussions with four process engineers that have experience in the implementation of such decision support techniques on large petrochemical plants (Alberts, 1997; Joubert and Theron, 1997; Nieuwoudt, 1998) showed that this is not always the case. A number of reasons were provided for the failure of projects that specifically implement artificial intelligence techniques for use in operational decision support. Those relevant to this dissertation are given below.

Only neural network or fuzzy logic approaches were used since the training algorithms of more intelligible models such as decision trees or crisp, "if...then..." type rules were found by the process engineers to be unable to meet the required accuracy levels. Furthermore, the algorithms that had been developed for generating decision trees or crisp rules were unsuitable for the particular applications concerned as they were predominantly designed for classification problems. The decision support systems that the process engineers had worked



on required regression methods that were capable of dealing with both continuous input and output data.

In the case where neural network or fuzzy logic approaches were used, it was found that the human process operators did not understand the models employed. This in turn meant that operators did not always follow the suggestions given to them by the relevant decision support model. This was especially true if the model made suggestions that were contrary to their experience of process operation. The operators did not understand how the model reasoned to produce such advice. This factor was especially prevalent if the operators had not been involved in the implementation and testing of the decision support system. In addition, the operators felt that the decision support system threatened their job security as the system was performing the same function as they were, viz. making changes to the operation of the chemical process to optimise some objective function, e.g. productivity. Furthermore, when an operational mishap occurred, the operators blamed the decision support model for causing the error in operation. Owing to the relatively unintelligible nature of the models that had been derived, the process engineers were unable to easily refute such claims. The above issues, together with the fact that the process engineers were unable to regularly maintain the decision support systems, resulted in the decision support systems being either ignored or switched off after two or three months of operation.

Model intelligibility is clearly a very important factor in determining the usefulness of a decision support system. This is especially true where the output or suggestions of the system need to be validated. This dissertation seeks to address two of the problems described above by proposing a new technique that induces fuzzy rules from chemical process data for use in a decision support system. These problems are the poor performance of algorithms that construct easily understood process models and the lack of intelligibility of models, such as trained neural networks, that perform well in terms of accuracy. The design objectives of the study are to create an algorithm that constructs models that are understandable and exhibit good accuracy. The particular constraint of this investigation is that model comprehensibility is seen as more important than obtaining the most accurate possible process model. Design decisions are therefore biased in favour of intelligible process models rather than model accuracy.

## 1.2 Aspects of Comprehensibility

Model intelligibility and comprehensibility is of primary concern in this dissertation. There are a number of different interpretations of these terms. In this dissertation the understanding of intelligibility and comprehensibility is based on the comprehensibility postulate of Michalski (1983).

*The results of computer induction should be symbolic descriptions of given entities, semantically and structurally similar to those a human expert might produce observing the same entities. Components of these descriptions should be comprehensible as single "chunks" of information, directly interpretable in natural language, and should relate quantitative and qualitative concepts in an integrated fashion.*

From the above description it is clear that models utilising natural language constructs to represent results are seen as more comprehensible than models that use other forms of representation. This is substantiated by comparative studies (Shavlik, et al., 1991; Donald, 1994; King, et al., 1995) between decision tree algorithms, methods that derive "if...then..." type rules, neural network techniques and statistical methods. The studies found that rule-based models that used natural language to describe learnt concepts were more intelligible than the neural network or statistical models that were considered. In other words rule-based models are thought to be significantly more intelligible than most neural network or functional forms of models.

In addition, discussions with the process engineers mentioned in section 1.1, brought to light that in many cases the process operators who use decision support systems have relatively little formal education (typically secondary level). These operators therefore struggle to use models that are described to them using even relatively simple mathematical concepts.

Therefore, owing to the better comprehensibility of rule-based models, this dissertation focuses on the different rule construction techniques that have been developed as a point of departure and uses the results of this analysis to develop an improved rule construction algorithm.

## 1.3 Rule-based Description of Chemical Processes

The second chapter of this dissertation describes a number of different rule construction techniques, enumerated below. The techniques that are described are seen as representative of the state-of-the-art in the various fields of research from which they stem. The different rule construction techniques that are considered are tree-based rule induction, rule induction by the covering approach, fuzzy rule modelling and rule extraction from neural networks. Emphasis is placed on the language used to represent the final set of learned rules and the search techniques that are used to derive the models.

The next part of the dissertation, chapter three, provides an evaluation of the techniques described in chapter two as well as those techniques that have not been discussed in detail but that are relevant to this investigation. The purpose of the evaluation is to draw attention to the overall limitations of the various algorithms that have been proposed in the different fields of research. Specifically, both the search strategies and the trained model representations of the different techniques are reviewed. In addition, historical trends in rule construction are discussed in terms of how they have affected the comprehensibility of the rules that are generated by the different techniques.

## 1.4 The Combinatorial Rule Assembler Algorithm

Based on the results of the above comparison a new algorithm, the Combinatorial Rule Assembler (CORA) algorithm, is proposed and described in chapter four. The CORA algorithm constructs fuzzy “if...then...” rules. The algorithm is primarily designed to model regression problems but it can also be used for classification problems. The first part of the chapter describes the Growing Neural Gas (Fritzke, 1994a) radial basis function network training method and the Reactive Tabu Search (Battiti and Tecchiolli, 1994a) combinatorial optimisation technique. These two methods are used as a basis for the CORA algorithm. Reasons are given why these particular algorithms were chosen above other possible candidates. The final section of chapter four describes the CORA algorithm in detail.

Chapter five and six evaluate the performance and properties of the CORA algorithm. The aim of the investigation is to determine whether the predictive performance and complexity of rule models constructed by the CORA algorithm are competitive with the results obtained by other techniques. Model complexity is seen as a quantitative measure of model comprehensibility and intelligibility. In addition, the contribution of the different components

that make up the CORA algorithm is evaluated. Finally, the influence of the training parameters of the CORA algorithm is also assessed.

Unfortunately implementations of only a few of the techniques described in chapters 2 and 3 are available to the author for extensive empirical testing. These are the CART set of decision tree algorithms (Breiman, et al., 1984) and the BEXA rule induction algorithm (Theron and Cloete, 1996). Two radial basis function techniques that construct 0<sup>th</sup>-order Sugeno fuzzy rules are also evaluated. The first is trained with the Growing Neural Gas algorithm and the second using *k*-means clustering (Moody and Darken, 1989).

In addition, the results obtained by three algorithms that do not construct rule-based models are also evaluated. These are the multilayer perceptron trained by the generalised delta rule (Rumelhart, et al., 1986; Zurada, 1992; Haykin, 1994), chosen as representative of neural network models, and the following two statistical methods. For regression problems multivariate linear regression (Seber, 1977) is evaluated. Finally, the multivariate adaptive regression splines (MARS) algorithm of Friedman (1991) is also included in the comparison. The MARS algorithm is chosen as representative of the state-of-the-art in statistical, nonparametric modelling of regression problems. These neural network and statistical methods are included in the comparison in order to provide a more global perspective on the performance of the CORA algorithm and other rule-based techniques.

## 1.5 The Modelling of a Chemical Process

Chapter 7 presents the application of the CORA algorithm to the modelling of a specific chemical reaction system. In particular, the problem presented to the CORA algorithm consists of the modelling and prediction of a continuous, nonlinear time series. The reaction system consists of three autocatalytic reactions that take place in an isothermal continuous flow stirred tank reactor (CSTR). Furthermore, the reaction system has been shown (Lynch, 1992) to exhibit chaotic phenomena. The data for this problem are generated synthetically. Noise is therefore added to the data to better approximate data acquisition in reality (where measurement noise, etc. would be present).

The purpose of the investigation performed in chapter 7 is to determine whether the CORA algorithm is capable of constructing models that are significantly (on a statistical basis) more accurate and less complex than existing model formats, based on a difficult chemical process modelling problem. (The presence of both chaotic phenomena and noise in the data makes the nonlinear time series prediction problem (in the author's opinion) difficult to solve.) The

models generated by the CORA algorithm are compared to the models generated by the same regression techniques that are evaluated in chapters 5 and 6. (Classification algorithms such as BEXA are not considered because the time series prediction problem is set as a regression problem and not as a classification problem.) Finally, a brief look is taken at the visual appearance of the fuzzy rules that are derived by the CORA algorithm.

## **1.6 Conclusions and Future Work**

Chapter 8 provides a summary of the results presented in this dissertation. In addition, conclusions regarding the work presented in this dissertation are also given. As is generally the case for work of this nature, as many (if not more) questions have been raised as have been answered during the development and evaluation of the CORA algorithm. The final part of chapter 8 therefore gives suggestions for possible future research that can be performed, based on the ideas encapsulated in the CORA algorithm.

# Chapter 2

## Existing Rule Construction Techniques

Research on the topic of rule construction has evolved extensively over the past few decades. There is a wealth of literature focussing on techniques or algorithms that derive “if...then...” type rules. This research comes under a number of different guises depending on the specific field of research. The more prominent fields are decision tree induction, rule induction by covering, fuzzy rule modelling and rule extraction from neural networks.

This chapter provides a description of a few of the popular or more advanced rule construction techniques that have been proposed in the different fields of research mentioned above. The techniques chosen are those that are thought to be representative of the general approach followed in each research field. Those aspects of the techniques that are relevant to the research undertaken for this dissertation are highlighted. In particular, emphasis is placed on describing the search strategies by which the rules are constructed and the different representations of the final set of learnt rules.

Crisp rule learning using genetic algorithms will not be explicitly discussed in this chapter. An extensive search of the relevant literature available to the author revealed that since roughly 1994 very little research has been published in the field of crisp rule learning using genetic algorithms. Rather, in the past few years most research using either standard genetic algorithms or the more recent evolutionary computation methods has concentrated on building fuzzy rules. For this reason only fuzzy rule learning using genetic algorithms will be described in this chapter (see section 2.3.5).

## 2.1 Describing Concepts with Rules

Two general types of rules will be considered in this dissertation. These are rules that are based upon propositional logic, defined as crisp rules, and rules that are based upon fuzzy logic, defined as fuzzy rules.

Crisp rules are typically of the form “If the reactor temperature is less than 348 °C and the feed material flow rate is between  $9.8 \text{ m}^3 \cdot \text{s}^{-1}$  and  $15.3 \text{ m}^3 \cdot \text{s}^{-1}$  then the product quality is 95.1 %”. In this example “reactor temperature” and “feed material flow rate” are attributes of a chemical reactor that are used to predict the quality of a particular product produced by the reactor. A possible fuzzy rule describing the same system can be “If the reactor temperature is *Low* or *Moderate* and the feed flow rate is *Generally High* then the product quality is *Very Good*”. Here fuzzy terms such as “*Low*” and “*Generally High*” determine the degree to which the fuzzy rule is applicable, given a set of crisp temperature and feed material flow rate values. The following two sections describe the similarities and differences of crisp and fuzzy rules in more detail.

### 2.1.1 Crisp Rules

Crisp rules consist of an antecedent part, or body, and a consequent. For classification rules the consequent is the predicted class. The consequent of a regression rule is a value or function that defines the predicted value of the rule. The body consists of a conjunction of antecedents. Each antecedent is usually a condition involving a single independent variable, or attribute. The various types of attributes that are used to build crisp rules are defined as follows. Categorical attributes take a finite set of unordered values. A Boolean attribute can either be true or false. Linear attributes, otherwise known as numerical attributes, have linearly ordered domains. For categorical attributes the simplest condition is an equality test. An example is “ $A_1$  is  $a_1$ ” where  $A_1$  is the attribute and  $a_1$  is the attribute value. More complex conditions allow negation and disjunction of attribute values. An example of disjunction is “ $A_1$  is  $a_1$  or  $a_2$ ”. For linear attributes the condition is inclusion within either a one-sided or two-sided interval. Examples of the latter form are “ $b_1 \leq B_1 \leq b_2$ ” or “ $B_1$  is between  $b_1$  and  $b_2$ ” where  $b_1$  and  $b_2$  represent linear attribute values of attribute  $B_1$ .

More advanced rule construction techniques (e.g. the OC1 system of Murthy (1996)) are able to build antecedents with conditions that use more than one linear attribute for the definition of a numerical interval. An example of one such condition is “ $5 \times A_1 + 3.2 \times B_1 \geq c_1$ ”. ( $c_1$  is an attribute value of data attribute  $C_1$ .) This condition is based on a linear combination of attributes. Some rule construction techniques, e.g. the TREPAN algorithm (Craven, 1996), are able to build rules



that can use  $p$ -of- $n$  expressions for the logical tests of the antecedent conditions. A  $p$ -of- $n$  expression is a Boolean expression that is specified by an integer threshold,  $p$ , and a set of  $n$  logical tests. The expression is satisfied when at least  $p$  of the  $n$  logical tests are satisfied. An example is “2-of- $\{A_1$  is  $a_1$ ,  $B_1$  is greater than  $b_1$ ,  $C_1$  is TRUE $\}$ ” where two of the three logical tests must hold for the antecedent condition to hold true. The  $n$  logical tests may be simple equality tests, or either one-sided or two-sided intervals on single attributes.

## 2.1.2 Fuzzy Rules

### 2.1.2.1 Definition of the Linguistic Variable

The definition of a linguistic variable is essential to the understanding of the format of a fuzzy rule and therefore will be presented first. A linguistic variable consists of a number of linguistic terms that are interpreted in terms of a base variable. A base variable is the same as the linear attribute defined for crisp rules. Each linguistic variable is fully characterised by the quintuple  $(v, T, X, g, m)$ .  $v$  is the name of the variable.  $T$  is the set of linguistic terms of  $v$  that refer to a base variable whose values range over a universal set  $X$ .  $g$  is a syntactic rule for generating the linguistic terms and  $m$  is a semantic rule that associates with each linguistic term  $t \in T$  its meaning  $m(t)$ , where  $m(t)$  denotes a fuzzy set in  $X$ . A fuzzy set  $m(t)$  defined on a universal set  $X$  is characterised by a membership function  $\mu_m(x)$  which takes on values in the interval  $[0,1]$ . The membership function determines the degree of similarity between an element of  $X$  and the given fuzzy subset. A membership value of 1 indicates complete similarity and a membership value of 0 indicates no similarity.

### 2.1.2.2 Types of Membership Functions

A number of different types of membership functions can be used. Five of the most common one-dimensional membership functions are described here. The membership functions considered are the triangular membership function, the trapezoidal membership function, the generalised bell membership function, the Gaussian membership function and the two-sided Gaussian membership function. The often-used multidimensional membership function, the multidimensional Gaussian membership function, will also be described. Klir and Juan (1995) and Jang, et al. (1997) provide more information on one-dimensional membership functions, such as the S-shaped membership function and the  $\pi$ -membership function, as well as a number of multidimensional membership functions.



The *triangular* membership function is specified by three parameters  $a$ ,  $b$  and  $c$  (with  $a < b < c$ ) that determine the  $x$  coordinates, on the universal set  $X$ , of the three corners of the membership function. The triangular membership function is defined by the expression

$$\text{triangle}(x; a, b, c) = \begin{cases} 0, & x \leq a \\ \frac{x-a}{b-a}, & a \leq x \leq b \\ \frac{c-x}{c-b}, & b \leq x \leq c \\ 0, & c \leq x \end{cases} \quad \text{Equation 2.1}$$

The *trapezoidal* membership function is specified by four parameters  $a$ ,  $b$ ,  $c$  and  $d$  (with  $a < b \leq c < d$ ). These parameters determine the  $x$  coordinates of the four corners of the underlying trapezoidal membership function. This membership function is described by equation 2.2.

$$\text{trapezoid}(x; a, b, c, d) = \begin{cases} 0, & x \leq a \\ \frac{x-a}{b-a}, & a \leq x \leq b \\ 1, & b \leq x \leq c \\ \frac{d-x}{d-c}, & c \leq x \leq d \\ 0, & d \leq x \end{cases} \quad \text{Equation 2.2}$$

Note that if  $b$  is equal to  $c$  then the trapezoidal membership function reduces to the triangular membership function. Note too that both the triangular and trapezoidal membership functions are not smooth at the corner points specified by the parameters. This makes complete, smooth differentiation impossible.

The *generalised bell* membership function, otherwise referred to as the *Cauchy* membership function, is also a smooth, nonlinear function that is specified by three parameters  $a$ ,  $b$  and  $c$  where  $b$  is usually positive. The formula for this membership function is given by equation 2.3.

$$\text{bell}(x; a, b, c) = \frac{1}{1 + \left| \frac{x-c}{a} \right|^{2b}} \quad \text{Equation 2.3}$$

The *Gaussian* membership function is specified by two parameters  $c$  and  $\sigma$  and is a symmetric, smooth, nonlinear function.  $c$  determines the centre of the Gaussian and  $\sigma$  represents the membership function's width. The formula for the Gaussian membership function is

$$\text{gaussian}(x; c, \sigma) = e^{-0.5 \left( \frac{x-c}{\sigma} \right)^2} \quad \text{Equation 2.4}$$

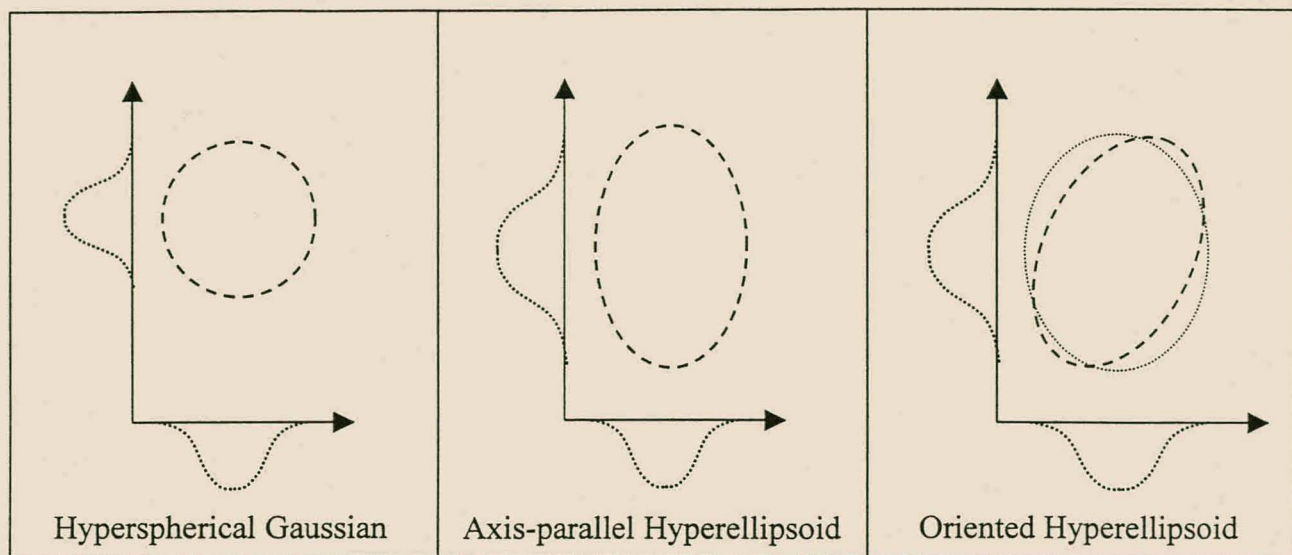
The *multidimensional Gaussian* membership function is a simple extension of this formula. The centre of the membership function is in this case specified in each dimension by the vector  $c$ . A general expression for the multidimensional Gaussian membership function is

$$\text{gaussian}(x; c, M) = e^{-\frac{1}{2}(x-c)^T M(x-c)} \quad \text{Equation 2.5}$$

$M$  is known as the metric of the Gaussian and stores the width information of the Gaussian. The metric is represented by a positive matrix  $A$ . The properties of  $A$  determine the shape of the Gaussian in hyperspace. More specifically, if  $A$  is diagonal with identical diagonal elements then the shape of the Gaussian will be a hypersphere. The width of the Gaussian in each dimension will therefore be identical.

If  $A$  is diagonal but the diagonal elements are not equal, then the Gaussian will form a hyperellipsoidal shape. The principal axes of the hyperellipsoid will be axis-parallel, viz. run parallel to the linguistic variable space axes. In this case the width of the Gaussian in each dimension may be different. Owing to the nature of the exponential function the one-dimensional components, or projections on the linguistic variable space axes, of both hyperspherical and axis-parallel, hyperellipsoidal Gaussians can easily be obtained. These one-dimensional components are unique and each the same as a one-dimensional Gaussian membership function.

If the matrix  $A$  contains off-diagonal elements the Gaussian will still form a hyperellipsoid but the principal axes of the hyperellipsoid will not be axis-parallel. In other words the principal axes of the hyperellipsoid will be described by some linear combination of the linguistic variables (Kosko, 1997). This means that the projections of such a hyperellipsoid on the original linguistic variable axes will not be unique. In other words, in contrast to the one-dimensional Gaussian membership functions obtained from an axis-parallel, hyperellipsoidal Gaussian, these projections will not uniquely describe the original oriented, hyperellipsoidal Gaussian.



**Figure 2.1 Hyperspherical and Hyperellipsoidal two-dimensional Gaussians**

The lefthand illustration presented in figure 2.1 is an example of a hyperspherical Gaussian in two dimensions (seen from above). The two-dimensional Gaussian contour plot depicted is projected on to the two attribute axes to form two, one-dimensional Gaussians. The picture in the middle of figure 2.1 illustrates a two-dimensional, axis-parallel hyperellipsoid with its one-dimensional Gaussian projection in each attribute dimension. The righthand picture presented in figure 2.1 depicts an oriented hyperellipsoid. Note that the one-dimensional Gaussian projections do not uniquely represent the original oriented hyperellipsoid. In other words, these one-dimensional Gaussians can come from more than one oriented hyperellipsoid, as illustrated by the third two-dimensional contour plot given in the picture.

The final one-dimensional membership function that will be considered here is the *two-sided Gaussian* membership function. The lefthand shoulder of this membership function is specified by  $c_1$  and  $\sigma_1$  and the righthand shoulder by  $c_2$  and  $\sigma_2$ . These parameters indicate the centre and width of the lefthand and righthand shoulders of the two-sided Gaussian, respectively. The formula for this membership function is given by equation 2.6.

$$two - sided\ gaussian(x; c_1, \sigma_1, c_2, \sigma_2) = \begin{cases} e^{-0.5 \left( \frac{x-c_1}{\sigma_1} \right)^2} & , x \leq c_1 \\ 1 & , c_1 \leq x \leq c_2 \\ e^{-0.5 \left( \frac{x-c_2}{\sigma_2} \right)^2} & , c_2 \leq x \end{cases} \quad \text{Equation 2.6}$$

### 2.1.2.3 Fuzzy Rule Format

As with a crisp rule, a fuzzy rule also consists of an antecedent part, or body, and a consequent. The body consists of a conjunction of antecedents. In simple fuzzy rules each antecedent is a condition involving a single linguistic variable. The condition is of the form “ $v$  is  $t$ ” where the degree of compatibility is determined by  $m(t)$ . In some cases the condition may be disjunctive, viz. have the form “ $v$  is  $t_1$  or  $t_2$ ”. Conjunction and disjunction are defined in terms of fuzzy logical operators. A number of different operators have been proposed. An example of a disjunction operator is the max operator (returns the maximum of two arguments) and of a conjunction operator the min operator (returns the minimum of two arguments). See Klir and Yuan (1995) for a detailed description of a number of different fuzzy operators and their properties. More complex rules have antecedents each with conditions that are based upon combinations of linguistic variables.

Fuzzy rules based on the Mamdani fuzzy model (Mamdani, 1975) each have both the antecedent part and the consequent defined using linguistic variables. The linguistic terms of these variables are respectively interpreted in terms of the attributes and the output of the data under consideration. Both conjunction and disjunction may occur in the antecedent part of a Mamdani fuzzy rule. Negation is usually not used in Mamdani rules. An example of a two input and single output Mamdani fuzzy rule is “If reactor temperature is *Hot* and reactor pressure is *High* or *Very High* then the reaction rate is *Fast*”.

The Sugeno class of fuzzy rules (Takagi and Sugeno, 1985; Sugeno and Kang, 1988) has an antecedent part of the same form as the Mamdani fuzzy rule model. Both conjunction and disjunction can occur. In contrast to the Mamdani fuzzy rule model, the consequent is some crisp function  $f$  of the base variables used in the antecedent part of the rule. If  $f$  is a constant the rule is called a 0<sup>th</sup>-order Sugeno rule. An example of such a rule is “If reactor temperature is *High* and feed flow rate is *Generally High* then waste gas production rate is  $17.3 \text{ m}^3 \cdot \text{h}^{-1}$ ”. If  $f$  is a 1<sup>st</sup>-order, linear function, the resulting rule is called a 1<sup>st</sup>-order Sugeno rule. An illustration of such a rule is “If reactor temperature is *High* and feed flow rate is *Generally High* then waste gas production rate is  $3x + 2.3y + 9 \text{ m}^3 \cdot \text{h}^{-1}$ ”.  $x$  and  $y$  represent “reactor temperature” and “feed flow rate” respectively. Higher order Sugeno rules are possible, depending on the choice of  $f$ .

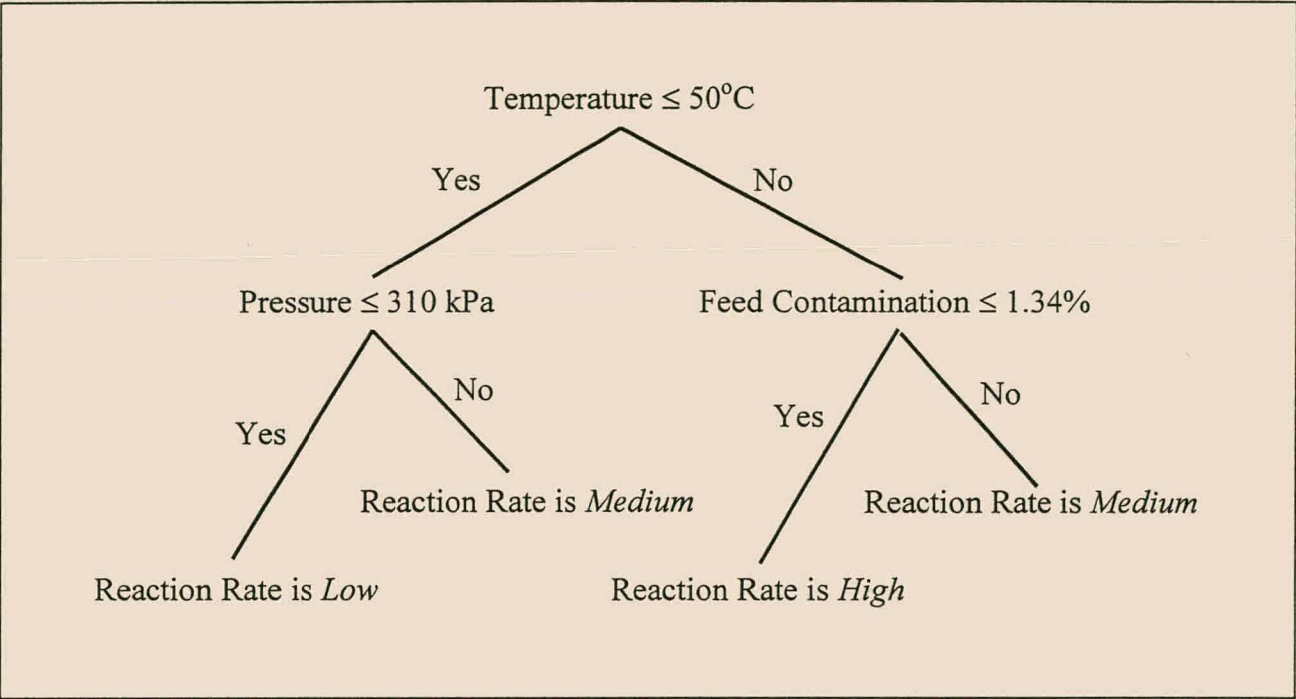


## 2.2 Crisp Rule Induction

Crisp rule induction deals with the construction of rules from data that have the crisp format described in section 2.1.1. There are two primary routes that are currently followed in the construction of such rules. These are the decision tree approach, described below, and the rule induction by covering approach, described in section 2.2.2.

### 2.2.1 Tree-based Rule Induction

#### 2.2.1.1 Decision Trees



**Figure 2.2 A Typical Classification Tree**

A decision tree (Figure 2.2) is a rooted, directed acyclic graph that consists of internal decision nodes and external leaf nodes connected by branches. Each decision node has an associated logical test that is used to decide which child node to visit next. In the case of binary decision trees each decision node has exactly two child nodes. The logical test of the decision node is based on one or more attributes of the data under consideration. A univariate decision tree contains logical tests that are based upon a single attribute whereas multivariate decision trees may contain logical tests that are functions of one or more attributes. The tree depicted in figure 2.2 is a univariate classification tree. A decision tree partitions the input space of the data set into a number of mutually exclusive regions. The decision nodes define the boundaries of each of these regions.

A leaf node has no child nodes and has a single label or value associated with it. Depending on the problem under consideration, the label or value of the leaf node characterises the values of the output of the data that fall within the given region. For classification trees each leaf node is assigned a label that indicates the predicted class. Decision trees used for regression problems are often called regression trees. For these trees a leaf node may have a constant value or be assigned some function of the attributes of the data. The constant value or function specifies the predicted output value of the leaf node.

Two of the most popular techniques with which univariate decision trees are derived from data are the C4.5 algorithm (Quinlan, 1993a) and the CART set of algorithms (Breiman, et al., 1984). The latter set of algorithms will be collectively called CART. C4.5 is applicable to classification problems and CART can be used for both classification and regression. These algorithms have been investigated in numerous empirical studies (e.g. King, et al., 1995; Gascuel, et al., 1998) and are commonly used as benchmarks for the analysis of new decision tree induction algorithms. For these reasons these two algorithms will be taken as representative of decision tree induction. The next two sections describe how these two algorithms build decision trees. Thereafter C4.5Rules, an algorithm that converts classification trees induced by C4.5 into “if...then...” rules, is described.

### 2.2.1.2 Growing Trees

Both C4.5 and CART induce decision trees based on a training set of exemplars in a divide-and-conquer fashion. The tree is grown by determining a succession of decision boundaries or splits that partition the training data into disjoint subsets. Starting from the root node that contains all the training data, an exhaustive search is done to find the split in the data that best reduces an output error measure. Once the best split is found the data are partitioned into two or more disjoint subsets, depending on the particular algorithm under consideration. CART induces strictly binary trees. C4.5 can for categorical attributes generate decision nodes that split the data into more than two disjoint subsets. For numerical attributes either algorithm can generate only binary splits. The subsets of data are represented by the same number of child nodes originating from the parent node (at the start this is the root node). The same splitting procedure is then applied to the data of each child node. This recursive procedure terminates when one or more stopping criteria are met.

A number of error measures can be used to find the best split. For classification trees two of the best known error measures are the entropy function (Quinlan, 1988) and the Gini diversity index (Breiman, et al., 1984). For regression trees the error measure used to find the best split for a

decision node is usually taken as the squared error, or residual, of a local model employed to fit the subset of data of the current decision node. If the local model is a constant, the value of the constant is simply the average value of the output values of the subset of data. If the local model is a linear function of one or more of the attributes then the model parameters are usually found using a least-squares method.

The leaf nodes of trees induced by C4.5 are assigned a class label. The label is the most frequent class of the data covered by the leaf node. CART has the same leaf node format for classification trees but for regression trees the leaf node is assigned the average output value of the exemplars covered by the given leaf node.

### **2.2.1.3 Tree Pruning**

Trees obtained by the above growing method are often large and complex. Problems of overfitting and over-specialisation towards the training data are often encountered. This is especially of concern if the set of training data is noisy or small. These two problems can result in a decision tree that generalises poorly on new, unseen data. The tree will therefore have suboptimal classification or regression predictive accuracy.

C4.5 prunes a classification tree using a form of reduced-error pruning. Pruning is done in a greedy fashion by replacing each decision node by either a leaf node or one of the decision node's branches. The algorithm first computes a confidence interval around the resubstitution error rate (Quinlan, 1993a) for a given decision node. The decision node is pruned if the resulting resubstitution error rate for the modified subtree is within a user-specified confidence interval of the unmodified subtree's error rate.

CART uses minimum cost-complexity pruning, otherwise known as weakest-subtree pruning. After the initial decision tree has been induced, the pruning process proceeds in two stages. In the first stage a series of increasingly smaller trees are built on the training data. See Breiman, et al. (1984) or Jang, et al. (1997) for more details on this procedure. In the second stage, one of these trees is chosen as the pruned tree, based on the prediction accuracy of the tree on a pruning data set. This pruning set is a portion of the original training data that is used exclusively for pruning.

### **2.2.1.4 Converting Classification Trees into Rules Using C4.5Rules**

The C4.5Rules (Quinlan, 1993a) algorithm converts an unpruned classification tree induced by C4.5 into a set of propositional rules. The algorithm works as follows. First, every path from the root node to a leaf node in the classification tree is converted into one initial rule. Second, each

rule is simplified in a greedy manner by removing conditions that do not significantly contribute to the ability of the rule to discriminate between its predicted class and the other class labels. Third, the entire simplified set of rules is examined and rules that do not contribute to the global accuracy of the rule set are removed. This rule simplification procedure can lead to the rules not being mutually exclusive anymore. The simplified set of rules may also not cover the entire input space. A final rule simplification step is therefore done where the set of rules is ordered to minimise false positive errors and a default class is chosen. A false positive error is where a training exemplar is covered by some subset of rules but the class of the exemplar is predicted incorrectly. The default class is used to predict the class of an exemplar that is not covered by any of the simplified rules.

The decision trees induced by CART can also be converted to rule format by taking each unique path from the root node to a leaf node in the decision tree as one rule. CART does not have any explicit mechanism to simplify these rules once they have been translated into propositional format.

### 2.2.2 Rule Induction by Covering

Two techniques that induce rules using a covering approach will be described in this section. They are the RISE algorithm (Domingos, 1994) and the BEXA algorithm (Theron, 1994). Both RISE and BEXA build crisp, classification rules. RISE is reported (Domingos, 1996e) to produce similar or better results in comparison to the CN2 set covering algorithm (Clark, 1989), the PEBLS instance-based learner (Cost and Salzberg, 1993) and C4.5 (Quinlan, 1993a). Furthermore, the author of RISE states that the algorithm can be seen as a unification of two widely used empirical rule learning approaches: rule induction and instance-based learning (Aha, et al., 1991; Cost and Salzberg, 1993). The algorithm learns rules in parallel in a specific-to-general fashion. BEXA is reported (Theron and Cloete, 1996) to produce rules of comparable accuracy but of greater simplicity, and therefore improved intelligibility, than those produced by CN2 and C4.5. In contrast to RISE, BEXA builds rules sequentially in a general-to-specific fashion.

Based on the reported good performance of these techniques, they are seen as indicative of the state-of-the-art in this research field. This, and the fact that they use significantly different training strategies to construct rules, will therefore provide an up to date overview of the rule induction by covering approach.



### 2.2.2.1 Building Rules with RISE

A rule induced by RISE has a body consisting of a conjunction of antecedents. Each condition involves only one attribute. For categorical attributes the condition is an equality test. For numeric attributes the condition is inclusion within a two-sided interval. Intervals may be degenerate. In other words, if for the two-sided interval " $a_i \leq A \leq b_i$ " it is the case that  $a_i = b_i$ , then the condition is written " $A = a_i$ ". Internal disjunction of rules is not supported. In each rule there is at most one condition involving each attribute, and in some cases there may be none.

RISE initiates rule building by assuming each training exemplar is a maximally specific rule. Each rule therefore has exactly one antecedent per attribute with the condition an equality test. The consequent of the rule is the exemplar's class. As mentioned above, rule construction proceeds in parallel in a specific-to-general fashion. Generalisation is repeatedly attempted for each rule. This is done by finding the exemplar with the same class as the rule that is nearest to the area covered by the rule. The rule is then minimally generalised to cover the exemplar. For categorical attributes this means dropping conditions that do not satisfy the new exemplar. For numerical attributes, where necessary, one or more intervals are simultaneously extended to include the attribute values of the exemplar at the interval border.

The change in the rule antecedent part is adopted permanently if the accuracy of the entire rule set on the training data does not deteriorate. This means that a more general rule is always chosen above one with the same accuracy but with a more complex antecedent part. If two rules become identical in the course of generalisation they are merged into a single rule. Generalisation stops when no rule can be further generalised without decreasing the overall accuracy of the existing rule set. In the worst case, no generalisations are accepted and the final rule set is the original training set of data.

### 2.2.2.2 Building Rules with BEXA

BEXA builds rules with antecedent parts each consisting of a conjunction of antecedents. Each condition involves only one attribute. For categorical attributes the condition is an equality test. For numeric attributes the condition is inclusion within a one-sided, e.g. " $E \leq e_i$ " or " $E \geq e_i$ ", or two-sided interval. In contrast to RISE, both internal disjunction and negation of the antecedents for categorical and numerical attributes is allowed.

Some definitions are required in order to be able to describe the rule construction method of BEXA. These are the following. For a given class, those training exemplars that belong to the class are called positive exemplars. The remaining exemplars are called negative exemplars.

General antecedents are those that contain few conditions and cover many positive exemplars. A rule is said to cover an exemplar if its antecedent part holds true for the exemplar. A consistent antecedent is one that covers no negative exemplars (with respect to the given rule's class). The set cover of a set  $B$  is defined as a set  $C$  of subsets of  $B$  such that their union equals  $B$ .  $C$  is an irredundant set cover of  $B$  if the deletion of any set  $C_i$  from  $C$  will cause the union of the remaining sets in  $C$  to be a proper subset of  $B$ .

Define the most general antecedent that exists for a given set of training exemplars as *mga*. Furthermore, let  $A_C$  denote the subset of consistent antecedents of  $A$  where  $A$  is the set of all  $VL_1$  antecedents that exist for a given learning problem.  $VL_1$  (Michalski, 1975) is a multiple-valued extension to propositional logic. The following property distinguishes between those antecedents that belong to  $A_M$ , the set of most general consistent antecedents, and those that are in  $A_C - A_M$ . Let  $c$  be an arbitrary antecedent and  $R = \{r_1, \dots, r_n\}$  be the subset of attribute values that are excluded from *mga* to obtain  $c$ . Then  $c \in A_M$  if and only if the set  $\{X_N(r_1), \dots, X_N(r_n)\}$  is an irredundant set cover of  $N$  where  $N$  is the set of negative exemplars.

BEXA uses a general-to-specific beam search to sequentially build rules. By beam search is meant that the beam-width best set of rules found thus far is specialised, rather than just the single best rule. The algorithm starts with a single general rule (defined above as *mga*) that covers all the training exemplars. The antecedent part of the rule is then selectively specialised by excluding attribute values. The aim of excluding attribute values is to generate a rule that covers the greatest possible number of positive exemplars and the least negative exemplars. For categorical attributes exclusion implies dropping attribute values and for numerical attributes it means excluding intervals of the given numerical attribute.

The subset of attribute values that is considered for exclusion is based on three restrictions, the prevent-empty-conjunctions restriction, the irredundancy restriction and the uncover-new-negatives restriction. The aim of these restrictions is to reduce the number of antecedents that must be considered for exclusion and thus decrease the overall learning time. Another aim is to direct specialisation effort at excluding those conjunctions that should in theory not form part of a good quality rule, i.e. a rule with both a simple antecedent part and high accuracy. The prevent-empty-conjunctions restriction ensures that no empty antecedents, i.e. those antecedents that cover no positive exemplars, are considered for exclusion. The second restriction is used to guide BEXA to construct antecedents in  $A_M$ . These antecedents are likely to contain few conditions and cover many positive exemplars and few negative exemplars. The uncover-new-negatives restriction ensures that each newly excluded attribute value uncovers at least one new negative

exemplar. A further significance test, the log-likelihood test (Clark and Boswell, 1991), may also be used to restrict the choice of rule specialisation to those antecedents that cover significant exemplars in the training set.

The Laplace accuracy estimate is used to determine which specialisation, of those that remain after the above prepruning restrictions and two further stop-growth tests (Theron, 1994) have been applied, is chosen to apply to the current rule. Specialisation of the current rule is halted when no specialisation can be found that does not lower the Laplace accuracy of the current rule to below that of the original training set of data. Thereafter the positive exemplars covered by the final rule are removed from the training data and a new rule is determined for the remaining data. Rule induction continues until all the positive exemplars in the training data are covered or the stop-growth tests terminate the generation of specialisations and therefore the induction of new rules. Rules are induced in this fashion for each of the different classes that exist in the training data. In this manner BEXA induces a set of unordered, classification rules.

BEXA employs a postpruning scheme (Quinlan, 1987) that prunes rules in two steps. The first step simplifies individual rules by deleting conditions that are insignificant according to Fisher's exact test. The second step discards those rules that do not reduce the accuracy of the entire rule set on the complete set of training data.

## 2.3 Fuzzy Rule Modelling

Fuzzy rule construction methods for fuzzy modelling purposes are diverse. They include fuzzy clustering methods, fuzzy decision tree induction, genetic algorithm techniques and fuzzy neural network approaches. A number of hybrid methods have also been proposed, for example the CANFIS algorithm (Jang, 1994; Jang, et al., 1997). The CANFIS algorithm combines CART (Breiman, et al., 1984) with the ANFIS neural network algorithm (Jang, 1993). CANFIS first performs rule structure identification using CART. Parameter optimisation is then performed using the ANFIS algorithm.

### 2.3.1 Evaluation of Publications Describing Fuzzy Methods

Unfortunately there is little published information regarding the relative merits of the different fuzzy modelling approaches that are currently followed. Where some form of comparison is provided, the evaluation is based upon one or two criteria such as prediction error on an unseen test set of data and the number of rules that are derived by the particular method. Few of the comparative results are evaluated using standard statistical tests such as the paired-sample t-test

(Sachs, 1984). It is therefore difficult to judge whether reported improvements by newer techniques over older ones are statistically significant.

Furthermore, little attention is paid to the relative intelligibility and complexity of the rule models that are generated. This makes it difficult to judge whether any increase in rule complexity is justified for the reported gain in prediction accuracy, etc. In particular, new techniques have often been proposed that obtain better prediction accuracy than older methods but achieve this by using less understandable fuzzy rule models. An example is the fuzzy rule model of Kim, et al. (1997). The method is reported to obtain better predictive accuracy on the Box-Jenkins gas furnace data (Box and Jenkins, 1970) than earlier methods. Inspection of the rules that are constructed show that the antecedent part of each derived rule consists of a nonaxis-parallel hyperellipsoid, i.e. multidimensional Gaussian. Such an antecedent part is more difficult to interpret in terms of the original attributes than for instance an axis-parallel hyperellipsoid. The reason for this is that for a nonaxis-parallel, or oriented, hyperellipsoid each principal axis is described by a linear combination of the original attributes of the training data. Each principal axis of an axis-parallel hyperellipsoid is described by just one original attribute.

In many cases the same set of data has been repeatedly examined, although in some cases not exclusively. Data that has been frequently examined is the Box-Jenkins gas furnace data, the Iris data set (Fisher, 1936) and the Mackey-Glass time series (Mackey and Glass, 1977). Tong (1978, 1980), Sugeno and Tanaka (1991), Sugeno and Yasukawa (1993), Nie (1995), Wang and Langari (1995, 1996a, 1996b), Joo, et al. (1997) and Kim, et al. (1997) have all studied the Box-Jenkins gas furnace data. The Iris data set has been examined by Simpson (1992), Ishibuchi, et al. (1994, 1995), Abe and Lan (1995), Hong and Lee (1996), Nozaki, et al. (1996), Ishibuchi, et al. (1997), Abe and Thawonmas (1997), Castro and Zurita (1997) and Russo (1998). Wang and Mendel (1992), Jang (1993), Cho and Wang (1996), Nie and Lee (1996), Lin and Lin (1997), Chen and Xi (1998) and Juang and Lin (1998) have examined the Mackey-Glass time series data.

The repeated investigation of algorithms on the same data makes it difficult to establish the true generalisation capabilities of a new fuzzy rule construction technique. This is because the statistical characteristics of the data are by now well known. This makes it hard to judge whether published improvements of a given rule construction technique are owing to an improved technique in general or because of improvements made to the technique based on the known statistical characteristics of the data under consideration.

### 2.3.2 Criteria Used to Choose Fuzzy Rule Construction Methods

The following criteria are therefore used to decide which algorithms to describe in this section. First, at least one representative technique from each of the classes of fuzzy rule construction methods must be described. The classes of methods considered are fuzzy clustering methods, fuzzy decision tree induction, genetic algorithm techniques and fuzzy neural networks. Second, only techniques that have been evaluated and compared to other techniques on two or more sets of data are considered. The reported accuracy performance should be comparable or better than the other techniques that were considered. Third, techniques that derive intelligible rules are chosen above ones that build less understandable rules. Rules that have antecedent parts that are based upon axis-parallel membership functions are considered more intelligible than rules with antecedent parts that are based upon nonaxis-parallel membership functions. The intelligibility of rule consequent parts is not taken into consideration at this stage.

The final criterion for techniques that qualify for consideration is that the techniques must be primarily designed for modelling purposes. Fuzzy rule construction for the express purpose of building fuzzy controllers (e.g. Tanaka and Sugeno, 1992; Kobayashi, et al., 1993; Karr and Gentry, 1993; Johnston, 1994; Linkens and Kandiah, 1996; Nie and Lee, 1997) are not considered. The reason for this is that the principal aims of rule-based fuzzy controller design are usually different from those of fuzzy rule modelling. That is, in addition to the common goal of good predictive performance, fuzzy controller design includes the consideration of issues such as model stability, robustness and completeness (Ying, 1994; Johansen, 1994).

### 2.3.3 Fuzzy Rule Construction Using a Clustering Approach

Two fuzzy rule construction techniques will be considered here. These are the ellipsoidal fuzzy classifier (Abe and Thawonmas, 1997) and the adaptive fuzzy inference system proposed by Chen and Xi (1998).

Abe and Thawonmas (1997) compare the ellipsoidal fuzzy classifier to three other classifiers. These are a three-layered neural network classifier, a fuzzy classifier that constructs crisp hyperbox regions in the input space and a fuzzy classifier that uses crisp polyhedron regions in the input space. The latter two classifiers respectively use the hyperbox or polyhedron regions to construct the antecedent parts of the fuzzy rules that are generated. For three of the four problems that were considered the proposed ellipsoidal classifier obtained classification accuracy on unseen data that was comparable to the results obtained by the other classifiers. The ellipsoidal classifier achieved slightly worse classification on the fourth problem.



Chen and Xi (1998) found that their proposed adaptive fuzzy inference system obtained better prediction accuracy in comparison to a self-organising, counter-propagation neural network (Nie, 1995) in two of the four problems considered. The algorithm achieved comparable prediction accuracy to two fuzzy rule construction techniques for the other two problems but used fewer rules to achieve the result. The two techniques that were considered are the partitioning technique of Wang and Mendel (1992) and the constructive neural network approach of Choi and Choi (1994).

The above two techniques of Abe and Thawonmas, and Chen and Xi, are seen as representative of the state-of-the-art in fuzzy rule construction methods that use the clustering approach. This assumption is based on the reported good results, the fact that both methods build fuzzy rules with axis-parallel membership functions and that these two techniques are seen by their respective authors as improvements on previous fuzzy rule construction methods.

### 2.3.3.1 The Ellipsoidal Fuzzy Classifier

A fuzzy rule constructed by the ellipsoidal fuzzy classifier has a body consisting of a conjunction of antecedents. Each condition involves only one linguistic variable and there is one condition for each attribute. In other words, each cluster, or fuzzy rule, is described in each input dimension by a single linguistic term. Internal disjunction of rules is not supported. The consequent part of the rule denotes the crisp class represented by the fuzzy cluster. Each linguistic term is represented by a membership function. Abe and Thawonmas (1997) chose multidimensional Gaussian functions to represent each ellipsoidal cluster. The membership function for each linguistic term is therefore equivalent to a one-dimensional Gaussian.

In the first stage of fuzzy rule construction the training data are clustered in an unsupervised manner using a novel clustering technique. Normal clustering methods such as the fuzzy  $c$ -means clustering algorithm (Bezdek, 1973) are not used because the authors claim that such methods do not have a mechanism to control the minimum amount of data that a cluster must have to justify its existence.

Rule construction starts by assigning a single cluster to each class present in the training data. A cluster must contain a user-defined minimum number of data ( $N_{min}$ ) for it to be allowed to exist. This is done to ensure that each cluster exhibits good generalisation capability. Thereafter a greedy, iterative procedure is followed. An arbitrary cluster with centre  $c_{ij}$  is selected that covers more than a user-defined maximum number of data ( $N_{max}$ ). Subsequently the algorithm attempts to split the data belonging to the cluster into two distinct subsets. The error measure used to

select the best split is based upon the total size of two hypothetical, crisp hyperboxes that would respectively cover the two subsets of data. The axis-parallel split chosen is the one that both runs through  $c_{ij}$ , in the particular dimension of concern, and that minimises the total size of the two hyperboxes. Splitting halts on a particular cluster when no split can be found that generates two clusters that each covers more than  $N_{min}$  number of data. The splitting procedure continues for all clusters until no cluster can be found that can be successfully split into two, based on the aforementioned “minimum number of data” stopping criterion.

The second stage of the algorithm converts the clusters that are generated in the first stage into fuzzy rules. First, the covariance matrix  $Q_{ij}$  is estimated for each cluster  $ij$ . If  $Q_{ij}$  is singular all the off-diagonal elements of  $Q_{ij}$  are set to zero so that  $Q_{ij}$  becomes regular. This ensures that the principal axes of the associated ellipsoidal region remain axis-parallel. The resultant membership functions are therefore functions of a single, original linguistic variable. The final step of the algorithm tunes the width of the ellipsoidal region of each fuzzy rule in order to maximise classification accuracy using a gradient descent method.

### 2.3.3.2 The Adaptive Fuzzy Inference System

In contrast to the ellipsoidal fuzzy classifier, the adaptive fuzzy inference system can be used for both regression and classification, although the authors do not investigate the latter property. The particular fuzzy rules that Chen and Xi consider are of the 1<sup>st</sup>-order Sugeno type described in section 2.1.2.3.

A fuzzy rule constructed by the adaptive fuzzy inference system consequently has a body of a conjunction of antecedents. Each condition involves one linguistic variable represented by a single linguistic term and there is one condition for each attribute. No internal disjunction of rules can occur. The membership functions chosen to characterise the linguistic terms of the antecedent part are triangular membership functions. The consequent is represented by a linear function of the input linguistic variables. A typical rule of this type may be “If the contaminant concentration is *High* and the flocculent concentration is *Low* then the water quality is  $f$ ”. Here “*High*” and “*Low*” represent linguistic terms for the respective linguistic variables.  $f$  is a linear function of the linguistic variables.

Fuzzy rule construction proceeds as follows. First, unsupervised, competitive learning takes place to find the positions of a user-specified number of clusters. Second, the cluster centres are frozen. The radius of each cluster or local region is then specified as the Euclidean distance between the given cluster centre and the exemplar that is furthest away from the cluster centre



but that also belongs to the cluster. Third, the parameters of the consequent function  $f$  of each cluster, or fuzzy rule, are determined by an iterative recursive least squares (RLS) procedure. Only those exemplars covered by a particular cluster are used to find the parameters for the consequent of that cluster. The iterative process halts when the consequent parameters have converged.

The final step of the fuzzy rule construction process is implemented to ensure that all the training exemplars are covered in the input space by at least one cluster, i.e. fuzzy rule. For each exemplar that is not covered by a fuzzy rule, the algorithm incrementally increases the radius of each cluster simultaneously until the exemplar is covered by the local region of at least one cluster.

### 2.3.4 The Induction of Fuzzy Decision Trees

In general two different methods are used to obtain fuzzy decision trees. The first method induces fuzzy decision trees in two steps. First, each attribute is fuzzified by definition of linguistic terms and corresponding membership functions for each attribute. Second, the fuzzy decision tree is induced using these linguistic terms and membership functions in the logical tests at each decision tree node. Publications that propose methods that follow this approach are Weber (1992a, 1992b), Yuan and Shaw (1995), Baldwin, et al. (1996), Ichihashi, et al. (1996) and Janikow (1998). In some cases both steps occur simultaneously. In other words, fuzzification occurs during decision tree induction itself. More specifically, decision nodes are fuzzified as they are created. This entails defining new linguistic terms and associated membership functions for each attribute as they are required and redefining the crisp conditions at the decision nodes as fuzzy conditions (e.g. Zeidler and Schlosser, 1995, 1996; Ittner, et al., 1997).

The second method typically follows a three-step approach. First, a crisp decision tree is constructed using ID3 (Quinlan, 1986), a precursor of the C4.5 algorithm (Quinlan, 1993a) or CART (Breiman, et al., 1984). Second, the crisp conditions at the decision nodes of the decision tree are fuzzified. In some cases a third step is performed to optimise the parameters of these membership functions. Consequent identification, i.e. leaf node labelling, is also sometimes done at this stage. Tani and Sakoda (1992), Jang (1994), Chi and Yan (1996) and Suárez and Lutsko (1998) all propose methods that use this general approach. Jang (1994) for instance uses a crisp decision tree induced by CART to define the antecedent structure of a fuzzy neural network. Membership functions are then defined for each decision node. The membership function parameters and consequent parameters are thereafter optimised by training the neural network.

None of the publications that were examined by the author met all of the criteria specified in section 2.3.2. In particular, none of the examined publications presented results comparing the performance of the proposed fuzzy decision tree techniques to other techniques on two or more data sets. It was therefore decided to waive this requirement and to choose a single algorithm that met the other criteria and that falls under the first general method described above. The reason for this latter specification is that the first general method has more support in terms of the number of articles published that focus on fuzzy decision tree induction. In addition, the aim of this section is to highlight the particular methodology of fuzzy decision tree induction itself and not of the other techniques that are used to create a hybrid technique. Hybrid techniques such as those grouped under the second general method will therefore not be considered here.

Yuan and Shaw (1995) proposed the technique that will be described here. They compare the format of the fuzzy decision tree that can be induced by their technique to that of the techniques proposed by Cios and Sztandera (1992), Tani and Sakoda (1992) and Weber (1992a, 1992b). The advantages of their approach over those mentioned above are that the algorithm can handle classification problems with both fuzzy attributes and fuzzy classes represented by linguistic terms. Both numerical and crisp, categorical attributes can be handled with the latter treated as a special case of fuzzy linguistic terms with zero fuzziness. The technique of Yuan and Shaw is therefore seen as an up to date and representative example of fuzzy decision tree induction.

#### **2.3.4.1 The Type of Fuzzy Rules Induced**

The algorithm proposed by Yuan and Shaw (1995) induces fuzzy classification trees. Such a tree can be directly translated into a set of “if...then...” fuzzy classification rules. The fuzzy rules that are induced are of the Mamdani type. The fuzzy rule body consists of a conjunction of antecedents. No internal disjunction can occur. Each antecedent contains a single condition that involves one linguistic variable represented by a single linguistic term. There is one condition for each attribute and in some cases there may be none. A typical rule is of the form “If the fuel temperature is *High* and the tank pressure is *Very High* then the rate of evaporation is *High*”. Here “fuel temperature”, “tank pressure” and “rate of evaporation” represent linguistic variables represented by the linguistic terms “*High*”, “*Very High*” and “*High*”, respectively.

#### **2.3.4.2 Fuzzy Classification Tree Induction and Rule Generation**

The induction of a fuzzy classification tree follows a two-step approach. First the crisp attributes and classes are fuzzified. This involves defining linguistic terms and associated membership functions for each attribute. Yuan and Shaw (1995) use triangular membership functions. The

centres of these membership functions are determined using Kohonen's self-organizing feature-map technique (Kohonen, 1982), a form of unsupervised clustering. The numbers of membership functions as well as the amount of overlapping between membership functions are user-defined.

Second, the fuzzy classification tree is induced in a manner similar to that of C4.5 (see section 2.2.1.2). The primary difference is that classification ambiguity (Yuan and Shaw, 1995) is used as a splitting criterion instead of an entropy-based criterion such as the gain ratio (Quinlan, 1988). Branching of a given decision node is terminated if a so-called truth level (Yuan and Shaw, 1995), calculated for the exemplars covered by the decision node, exceeds a user-defined threshold. In this case the decision node is converted into a leaf node. The class of the leaf node is the one with the greatest truth level. If the truth level does not exceed the specified threshold, the decision node branches to form at least two child nodes. The algorithm is not restricted to binary splits, i.e. multiway splits can also be formed during tree construction. Node splitting continues for all newly generated decision nodes until no further growth is possible.

Fuzzy rules are extracted from the fuzzy classification tree as follows. First, each path from the root node to a leaf node is converted to a fuzzy rule. The rule antecedent part represents the conditions on the attributes encountered when passing from the root to the leaf node. The rule consequent is the leaf node class that has the highest truth level. A greedy rule simplification procedure is then performed. For each rule in turn conditions from the antecedent part of the rule are dropped one at a time. The truth level is determined for each simplified rule that is formed in such a manner. All the conditions that can be removed without decreasing the truth level of the simplified rule below a user-defined threshold are found. From these, the one that maximises the truth level of the simplified rule is removed permanently. This process continues for each rule until no further simplification is possible without decreasing the truth levels of the rules below the user-defined threshold.

## **2.3.5 Fuzzy Modelling using Genetic Algorithms**

### **2.3.5.1 The Genetic Algorithm**

A genetic algorithm is a general-purpose, derivative-free, stochastic optimisation tool (Holland, 1975; Goldberg, 1989). The algorithm is based loosely on the concepts of natural selection and evolutionary processes that were proposed in Darwin's evolutionary theory (Darwin, 1859).

The genetic algorithm encodes potential solutions to the optimisation problem in the form of so-called chromosomes. In terms of fuzzy rule modelling a chromosome can, for example, encode the structure (number of membership functions, etc.) and parameters of both the antecedent body

and consequent of each rule of a set of fuzzy rules. Each chromosome, or individual, has an associated “fitness” value determined using a predefined objective function. An example of such an objective function is the prediction accuracy of the individual on a set of training exemplars.

The genetic algorithm operates on a population of individuals. The optimisation procedure starts by determining the fitness of each individual in the current population. Based on these results a set of individuals is selected according to some predefined selection scheme (Goldberg, 1989; Back and Hoffmeister, 1991). Genetic operators are then applied to these selected individuals in order to evolve improved solutions to the problem. Two commonly used genetic operators are crossover and mutation.

The crossover operator is a mechanism, similar to sexual reproduction, whereby genetic material is redistributed between different individuals. It is applied to two parent individuals and generates two new child chromosomes. These child chromosomes are constructed by swapping parts of the parent chromosomes to form the two new individuals. The aim of the crossover operator is to construct child individuals that have combinations of the characteristics of the parent individuals. A child chromosome may therefore outperform either or both of its parent chromosomes if it obtains the “good” genetic traits of its parents.

The mutation operator emulates mutation in living species. It is applied to single chromosomes with the aim of generating new genetic characteristics. In addition, it can prevent the population from converging and stagnating at some local optima (Jang, et al., 1997). Mutation is implemented by randomly changing individual parts of the chromosome and thus creating new genetic material.

After these genetic operators have been applied to the selected set of individuals, the fitness of each of the newly created individuals is determined. All the individuals, both new and old, are then ranked in terms of fitness and a subset of these is selected to form a new population of individuals. In other words all the chromosomes compete, in terms of fitness, in order to survive to the next generation. Individuals with above-average fitness will survive to the next generation whereas below-average performing individuals will be discarded. In most cases the population size is kept constant and therefore the population-size best individuals will form the next generation. This sequential creation of new generations of individuals continues until one or more stopping criteria are met.

### 2.3.5.2 Choice of Genetic Algorithm Technique to Discuss

Genetic algorithms have been used extensively to generate fuzzy rules. This includes the identification of the structure, parameters and number of fuzzy rules required for good performance. Basic techniques assume that some of these aspects are predefined. For example, Ishibuchi, et al. (1994, 1995) and Ishibuchi, et al. (1997) propose methods that work on a predefined fuzzy rule structure. The genetic algorithm is used to determine the best subset of fuzzy rules from the set of predetermined rules. Karr and Gentry (1993) use a genetic algorithm to optimise the parameters of membership functions of a fixed number of fuzzy rules. More advanced techniques have used the genetic algorithm to find both the structure and parameters of a predefined number of fuzzy rules (e.g. Park, et al., 1994).

Apart from the five articles mentioned in the previous paragraph, six other publications (Yuan and Zhuang, 1996; Jäkel, 1997; Joo, et al., 1997; Pokorný, et al., 1997; Wong and Lin, 1997; Russo, 1998) were examined to find representative, genetic algorithm-based, fuzzy rule construction techniques. The investigation revealed that few authors (Jäkel, 1997; Russo, 1998) evaluate their respective techniques on two or more nontrivial, modelling problems.

Jäkel (1997) uses a genetic algorithm for antecedent structure identification and determines parameters using a least squares or nonlinear least-squares approach. Jäkel models a five-dimensional, nonlinear function as well as the Box-Jenkins data set (Box and Jenkins, 1970) but does not compare the results obtained by his algorithm with any other methods. Russo (1998) performs fuzzy rule structure identification as well as determining the number of fuzzy rules with an advanced genetic algorithm and parameter identification with a backpropagation multilayer perceptron. Russo compares his hybrid algorithm with a number of other techniques on two modelling problems and one control problem. Furthermore, the author gives references to six other fields (e.g. pharmacology (Santagati, et al., 1995)) where the algorithm has been applied.

The technique that is described in this section is the FuGeNeSys algorithm proposed by Russo (1998). The reason for this choice is that Russo (1998) compares the results obtained by the proposed technique to other methods on more than two problems. In addition, the author mentions a number of other fields in which the technique has been applied. Furthermore, the genetic algorithm is used for fuzzy rule structure identification and determining the number of fuzzy rules, and not for parameter identification. In the opinion of the author of this dissertation genetic algorithms are best suited to combinatorial problems such as fuzzy rule structure identification. A number of other powerful and efficient techniques already exist for determining



the optimal or near-optimal parameters of especially linear functions but also nonlinear functions and fuzzy rules. It was therefore decided to concentrate on a technique that uses a genetic algorithm for structure identification rather than for parameter identification or both.

### 2.3.5.3 Fuzzy Rule Construction Using FuGeNeSys

FuGeNeSys builds 0<sup>th</sup>-order Sugeno-type fuzzy rules. The rules are purely conjunctive, i.e. no internal disjunction can occur. Furthermore, the antecedent part of each rule contains at most one condition for each attribute and for some attributes there may be none. The linguistic terms used in each of these conditions are each represented by a one-dimensional Gaussian membership function. The consequent of each rule is a numerical constant in the case of a regression problem and a class label in the case of a classification problem.

FuGeNeSys implements a continuous, apomictic, fine-grain genetic algorithm. Such a genetic algorithm does not allow crossover between any random two individuals of the current population. The individual selection scheme is likewise limited. Rather, in such an evolution strategy a population is divided into a number of subpopulations often referred to as demes. Individuals are first placed on a rectangular, planar grid. Selection and crossover are then localised within an area with a user-defined radius on this planar grid. Isolation of demes is a direct consequence of the distance between them.

The fitness-proportional selection scheme used by FuGeNeSys is static, preservative, elitist and steady state. In fitness-proportional, static selection the probability of an individual being selected is proportional to its fitness. Preservative selection guarantees that each individual will have a non-zero probability of being selected for crossover and mutation. Elitist selection guarantees that the fittest individual will always be selected. Steady-state selection indicates that parent individuals are selected in a stepwise fashion and the offspring created by crossover and mutation are gradually introduced into the current population. In contrast to the generational selection scheme described in section 2.3.5.1, a set of parents is determined in a once-off fashion for the current population before any offspring are generated.

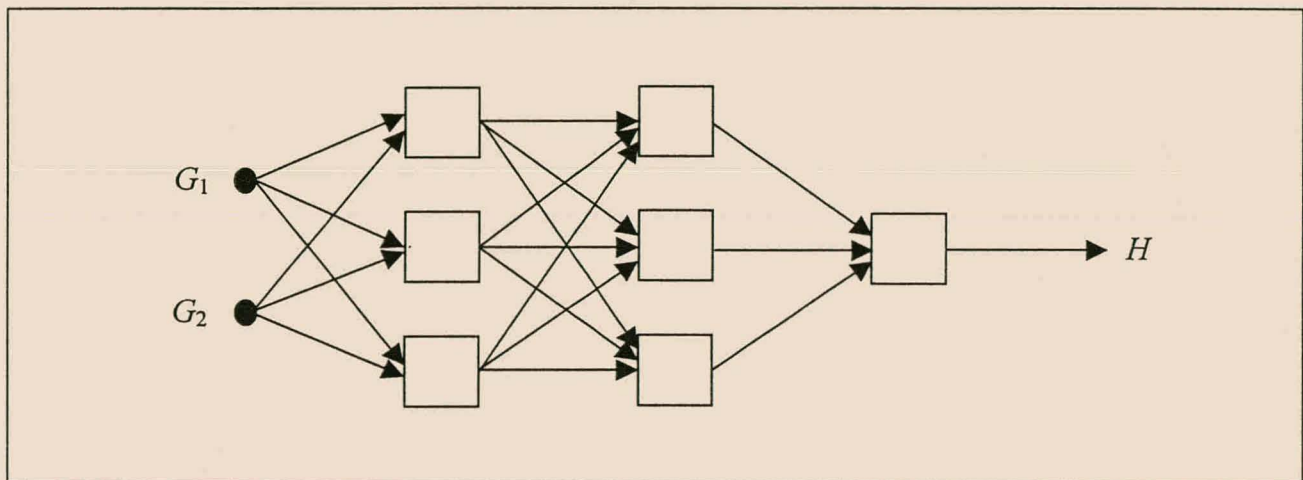
A further hill-climbing operator is used by FuGeNeSys for parameter optimisation. This operator is implemented during the evolution process whenever an individual is generated with a fitness value greater than the best one obtained so far. Note that each individual represents a complete set of fuzzy rules. First, the individual is transformed into an equivalent neural network. Second, the neural network is trained using a backpropagation procedure. Russo (1998) states that in this phase some neurons of the network, viz. fuzzy rules, may be deleted but this operation is not

described in any detail. This property will therefore be ignored in this dissertation. Third, the trained neural network is transformed back into a genetic individual. Finally, this new individual replaces the original individual in the population that it was initialised from.

## 2.3.6 Neural Network Methods

### 2.3.6.1 Neural Network Structure

This subsection provides a brief description of the typical structure of feedforward neural networks. Particular attention is paid to radial basis function neural networks. Refer to Haykin (1994) or Pham (1995) for information regarding these and other types of neural networks (e.g. recurrent neural networks) and the techniques that can be used to train them.



**Figure 2.3 Graphical Representation of a Neural Network**

In general, a feedforward neural network consists of a number of neural nodes that are grouped into a sequence of neural layers. Figure 2.3 gives a graphical depiction of a feedforward neural network consisting of four layers. There are two inputs ( $G_1$  and  $G_2$ ) and one output ( $H$ ). The network consists of an input layer (the solid bullets in figure 2.3), an output layer and any number of hidden layers between these two layers. Each neural node (squares in figure 2.3) represents a parameterised function that performs either a linear or nonlinear transformation of its input signals and returns a single output activation. The activations of the nodes in the input layer represent attribute values of exemplars in the training data. Typically a single input node represents a numeric attribute whereas a  $n$ -valued, categorical attribute is represented by  $n$  input nodes. For single output regression problems the output layer contains a single node and for multiple outputs more than one node is used. For classification problems the output class may either be encoded using  $n$  output nodes to represent  $n$  classes, or by using only a single output node. Hidden layers are added to the neural network to facilitate network learning by



transformation of the input space. Finally, network layers are connected via unidirectional, weighted connections (the arrows in figure 2.3) that specify the propagation of node outputs to nodes in succeeding layers. The weights of these connections are adjustable.

A radial basis function neural network (Broomhead and Lowe, 1988; Moody and Darken, 1989; Poggio and Girosio, 1990) is a particular type of feedforward neural network that typically consists of three layers: an input layer, a single hidden layer and an output layer. The network is usually fully connected, in other words each node in a preceding layer is connected to each node in the succeeding layer. Each neural node in the hidden layer is represented by a parameterised kernel function. This kernel function is often chosen to be a multidimensional Gaussian, such as the type described in section 2.1.2.2.

Only the connections between the hidden layer and output layer are weighted. Earlier radial basis function networks used a constant term to represent these weights. More advanced radial basis function networks (e.g. Langari, et al., 1997) use linear functions of the original input variables, or attributes, as weights. The one or more neural nodes in the output layer are typically summation nodes that each computes and returns the linear, weighted sum of all the outputs of the nodes of the hidden layer. More sophisticated output nodes each return the weighted average (Jang, et al., 1997) of the input signals to the given output node.

### **2.3.6.2 Equivalence Between Radial Basis Function Networks and a Set of Fuzzy Rules**

Before the neural network-based fuzzy rule construction techniques are described in more detail, the conditions under which a radial basis function network and a set of fuzzy rules, or fuzzy inference system, are equivalent will be summarised. If these conditions are met then a radial basis function network can be directly translated into a fuzzy inference system. The fuzzy inference system will perform in exactly the same manner as the trained radial basis function network. The conditions are (Hartman, et al., 1990; Park and Sandberg, 1991; Jang and Sun, 1993):

- The number of nodes in the hidden layer of the radial basis function network must be equal to the number of fuzzy “if...then...” rules. In other words, each radial basis function must be represented by an equivalent fuzzy rule.
- Each radial basis function of the radial basis function network must be equal to a multidimensional, composite membership function of the antecedent part of a fuzzy rule in the fuzzy inference system, i.e. set of fuzzy rules. This can be achieved by for instance choosing multidimensional Gaussian radial

basis functions. A multidimensional Gaussian can be decomposed into a set of one-dimensional Gaussian membership functions with the same width and relevant centre coordinate in each dimension as the original radial basis function. These membership functions form the antecedent part of the “if...then...” fuzzy rule.

- The firing strength of a fuzzy rule must be obtained in the same manner as the activation of the equivalent radial basis function in the radial basis function network’s hidden layer. This can be achieved by again choosing multidimensional Gaussian radial basis functions. The activation of a multidimensional Gaussian with respect to an input signal is the product of its activation in each dimension. The firing strength of the equivalent fuzzy rule can be obtained by using the product rule on the so-called truth values of the one-dimensional Gaussian membership functions in the antecedent part of the fuzzy rule.
- The consequent of a fuzzy rule should be identical to the weight of the corresponding radial basis function to the radial basis function network output node. That is, the same constant terms or linear response functions of the radial basis function network should be used for the consequents of the equivalent 0<sup>th</sup>-order Sugeno fuzzy rules or 1<sup>st</sup>-order Sugeno fuzzy rules, respectively.

### 2.3.6.3 Fuzzy Rule Construction Using Neural Networks

Numerous algorithms have been proposed for constructing fuzzy rules with the aid of neural networks. This section focuses on three general methods that have been widely studied, viz. fuzzy inference neural networks, radial basis function neural networks and the fuzzy ARTMAP class of algorithms.

Examples of the fuzzy inference network approach include the fuzzy min-max neural network (Simpson, 1992, 1993), the adaptive fuzzy inference system (Jang, 1993; Sun, 1994; Lotfi and Tsoi, 1996) and the hybrid CANFIS system (Jang, et al., 1997). CANFIS combines CART with an adaptive fuzzy inference system. Wang, et al. (1997) propose a fuzzy neural network that constructs fuzzy rules using predefined membership functions. The fuzzy inference network of Cai and Kwan (1998) and the self-constructing fuzzy inference network (SONFIN) algorithm of

Juang and Lin (1998) both automatically determine the number of fuzzy rules as well as the optimal rule parameters.

The radial basis function network approach has been studied by a number of authors. Hwang and Bang (1997) use a clustering algorithm to determine the position of the radial basis functions. Fritzke (1997) incrementally constructs radial basis function networks in a supervised fashion using the growing neural gas training method. Langari, et al. (1997) replace the constant weights to the output layer of a radial basis function network with regression weights. Yingwei, et al. (1997) combine the resource-allocating network of Platt (1991) with a pruning strategy in order to obtain a radial basis function network with a minimal hidden layer topology. The radial basis function network of Karayiannis and Mi (1997) automatically determines the correct number of radial basis functions by selectively splitting the existing radial basis functions of a radial basis function network. Billings and Hong (1998) use orthogonal least squares to select the most important input nodes as well as the most important regressors, or radial basis functions, of a radial basis function network. Their method does not directly generate fuzzy rules and will therefore not be considered further here.

Two fuzzy neural network methods, taken from those mentioned above, will be described in more detail here. These are the self-constructing fuzzy inference network (SONFIN) (Juang and Lin, 1998) and the growing radial basis function (GRBF) neural network algorithm (Karayiannis and Mi, 1997). As mentioned before SONFIN performs both structure identification as well as parameter identification. More specifically, SONFIN is able to automatically determine the number of fuzzy rules as well as the antecedent and consequent structure for each fuzzy rule. Parameter identification involves obtaining the parameters for both the antecedent part and consequent of each fuzzy rule.

Juang and Lin (1998) analyse the performance of SONFIN on five different problems, viz. identification of a dynamical system, nonlinear channel equalisation, water bath temperature control, chaotic time series prediction and noisy speech recognition. Only the results obtained by SONFIN on the chaotic time series problem are compared to those obtained by five other algorithms. These include FALCON-ART (Lin and Lin, 1997) and the adaptive vector quantization algorithms of Kosko (1992). For the chaotic time series problem SONFIN obtains either better accuracy than the other algorithms that were considered or achieves comparable accuracy but with a significantly smaller fuzzy model, i.e. fewer fuzzy rules.

The GRBF network (Karayiannis and Mi, 1997) is able to determine the optimal number of fuzzy rules as well as the structure and optimal parameters for both the antecedent part and

consequent of each fuzzy rule. Various forms of the algorithm that use different learning methods are compared to each other as well as to conventional feedforward neural networks on a two-dimensional vowel classification problem (Lippmann, 1989; Ng and Lippmann, 1991). The classification accuracy of the GRBF network was found to be worse than that of a single hidden layer feedforward network with backpropagation training but the GRBF network trained significantly faster. Different forms of the GRBF algorithm are also compared to each other and two conventional radial basis function networks on the Iris data set (Fisher, 1936). For this problem the trained GRBF network exhibited better classification accuracy than the conventional radial basis function networks that were considered.

A third fuzzy neural network architecture that will be studied is fuzzy ARTMAP (Carpenter, et al., 1992). Fuzzy ARTMAP is based on the ARTMAP class of neural network architectures developed on the basis of the Adaptive Resonance Theory (Carpenter, et al., 1991). The algorithm builds both fuzzy classification rules and fuzzy regression rules in an incremental, supervised fashion. Fuzzy ARTMAP is able to perform structure and parameter identification of an automatically determined number of fuzzy rules. Carpenter, et al. (1992) compare fuzzy ARTMAP with a multilayer perceptron with backpropagation training on the circle-in-the-square problem as well as the double spiral problem. For both these problems fuzzy ARTMAP significantly outperformed the multilayer perceptron on unseen test data. Furthermore, the fuzzy ARTMAP achieved significantly better test accuracy than a genetic algorithm system (Frey and Slate, 1991) on a letter image recognition problem. Hamker and Heinke (1997) compare fuzzy ARTMAP to the Growing Neural Gas algorithm (Fritzke, 1995a) and the Growing Cell Structures algorithm (Fritzke, 1994b). Fuzzy ARTMAP obtained significantly better results than the latter two algorithms on one of four problems. In contrast, fuzzy ARTMAP achieved similar or significantly worse results on the other three data sets that were considered.

The comparative results obtained for either SONFIN or the GRBF network are few, and in one case poor (the GRBF network on the vowel classification problem). Fuzzy ARTMAP has been compared to a significant number of different algorithms but the results do not indicate a clear superiority of the technique over all of the other algorithms that were considered. On the other hand, the structure and parameter identification capabilities of these three techniques are on the whole more sophisticated than those of the other techniques mentioned above. Therefore, in the author's opinion, a description of these three techniques will provide a good overview of the current state-of-the-art in neural network fuzzy modelling.

### 2.3.6.4 The Self-constructing Fuzzy Inference Network (SONFIN)

The fuzzy rules that are built by the SONFIN technique are of the Takagi-Sugeno-Kang (TSK) (Sugeno and Tanaka, 1991) form. The antecedent part of each rule consists of a conjunction of antecedents and there is exactly one condition for each attribute. No internal disjunction can occur. One-dimensional Gaussian membership functions are used to represent the linguistic terms used in each of these conditions. The consequent of each rule consists of a single linguistic term, or Gaussian membership function, based on the output linguistic variable as well as a number of linear terms. In some cases the consequent of more than one fuzzy rule may be identical.

An example of a fuzzy rule developed by SONFIN is “If the reactor temperature is *Moderately High* and the reactor pressure is *High* then the product production rate is  $Good + 5.4u + 3.2w$ ”.  $u$  and  $w$  represent the two base variables upon which the linguistic variables “reactor temperature” and “reactor pressure” are based. The consequent part of this rule therefore consists of both a fuzzy term as well as two crisp linear terms. All three terms need to be evaluated to determine the final output value of the rule.

The basic SONFIN constructs the antecedent parts of fuzzy rules using hyperellipsoidal, multidimensional Gaussians. In other words a fuzzy rule corresponds to a hyperellipsoidal cluster in the input space. The principal axes of these hyperellipsoids are axis-parallel and therefore the membership functions that represent the projections of a given multidimensional Gaussian are functions of a single linguistic variable. In contrast, the advanced form of SONFIN is capable of constructing oriented hyperellipsoids. That is, the principal axes of these hyperellipsoids are not axis-parallel. Membership functions that represent the projections of these hyperellipsoids are therefore functions of two or more of the original linguistic variables.

SONFIN is designed for online training, i.e. the fuzzy inference network is adapted in a sequential fashion whenever it is presented with new training data. Training of the three-layered network occurs in four steps. Upon the presentation of a new training exemplar SONFIN first determines whether the current set of fuzzy rules adequately covers the new exemplar in the input space. If the test result fails to meet a user-defined threshold a new hyperellipsoidal Gaussian that represents a new fuzzy rule is created. The multidimensional Gaussian is then decomposed into its corresponding one-dimensional Gaussians, or membership functions. In order to reduce the number of membership functions generated for each linguistic variable, and to avoid creating highly similar membership functions, the similarity in each dimension between the newly formed membership functions and each of the existing of membership functions of the



entire set of rules is calculated. A similarity measure proposed by Lin and Lee (1994) is used for this purpose. The membership functions of the new fuzzy rule that are highly similar to ones that have been generated for other fuzzy rules are replaced with the existing membership functions.

The next two training steps determine the consequent structure of the new fuzzy rule and also update the consequent structure of each existing rule. In short, a singleton value is first assigned to the new fuzzy rule using a clustering method. Thereafter, additional linear terms of the most significant input variables, i.e. attributes, are added to the rule consequent. This step occurs for each fuzzy rule in turn. A projection-based correlation measure algorithm is used to determine which terms are significant and need to be added to a given rule consequent.

The final step of SONFIN training uses either the least mean squares algorithm or the recursive least squares algorithm to determine the optimal consequent parameters and backpropagation training to optimise the antecedent part parameters of the entire set of rules.

### **2.3.6.5 The Growing Radial Basis Function Neural Network**

The antecedent part of each of the fuzzy rules constructed by the growing radial basis function (GRBF) network (Karayiannis and Mi, 1997) is represented by a hyperspherical, multidimensional Gaussian. As before the multidimensional Gaussian is factorised into its one-dimensional components to form one-dimensional membership functions. The antecedent part of a rule therefore comprises of a conjunction of antecedents and there is exactly one condition for each attribute. No internal disjunction occurs. The consequent of each fuzzy rule is a numerical constant. The fuzzy rules are therefore of the 0<sup>th</sup>-order Sugeno type.

Training of the GRBF network is initiated by creating two radial basis functions, viz. multidimensional Gaussians, in the empty hidden layer of the network. Thereafter, both the locations and widths of all the radial basis functions are updated. Karayiannis and Mi describe a number of different methods that can be used for this training step, including a novel class-conditional variance technique. The next training step updates the weights to the output layer of the radial basis function network using either recursive least squares or gradient descent learning.

If, after one such training cycle, the performance of the network is unsatisfactory, another radial basis function is added to the hidden layer. This is achieved by implementing a so-called splitting procedure whereby a single radial basis function is first selected, based on either a feedback or purity splitting criterion. The former criterion selects the radial basis function responsible for the highest prediction error for the splitting procedure. The latter criterion selects

the radial basis function that covers the least pure (with respect to the data class distribution) cluster of training data.

The parameters of both the new radial basis function and the chosen radial basis function are updated based upon the current parameter values of the chosen radial basis function. Cyclical training and addition of new radial basis functions to the network's hidden layer continues until a stopping criterion is met. The stopping criterion employs a validation set of data to determine when best to stop training.

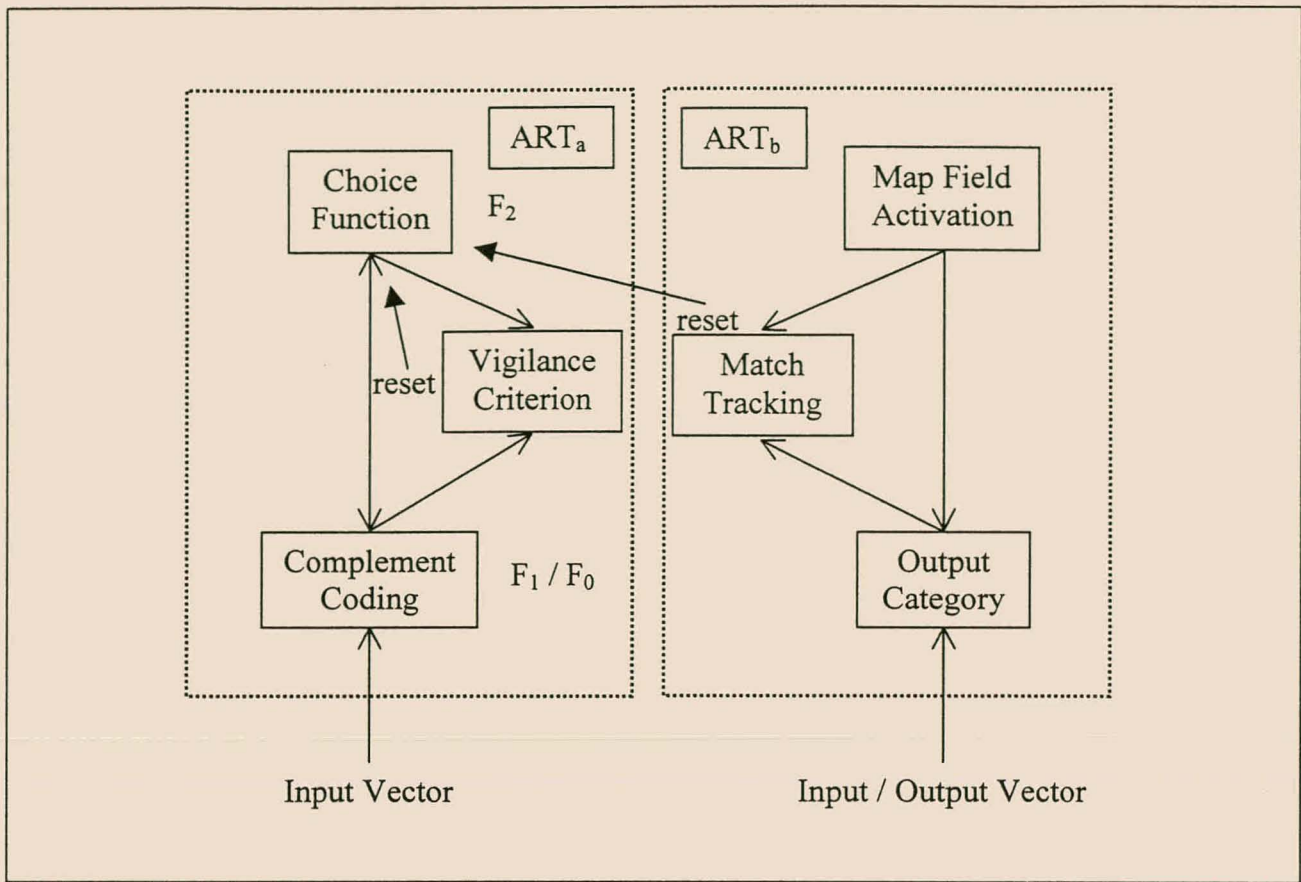
### 2.3.6.6 Fuzzy ARTMAP

The general structure of fuzzy ARTMAP is illustrated in figure 2.4. Each fuzzy ARTMAP contains a pair of adaptive resonance theory modules ( $ART_a$  and  $ART_b$ ). These modules are used to create stable recognition categories, i.e. fuzzy rule consequents, from a set of training exemplars. During supervised classification learning,  $ART_a$  is fed the attribute part, or input vector, of each of the training exemplars.  $ART_b$  is fed each training exemplar in its entirety, otherwise known as an input / output vector. An associative learning network and an internal controller link the two ART modules. The latter is designed to create the minimal number of  $ART_a$  recognition categories required to meet a user-defined accuracy criterion. It achieves this through a minimax learning rule (Carpenter, et al., 1992) that conjointly minimises predictive error and maximises predictive generalisation.

Each ART network contains a layer,  $F_0$ , of nodes that represent the attribute values of the current training exemplar. The  $F_1$  layer receives bottom-up input from  $F_0$  and top-down input from layer  $F_2$ . The  $F_2$  layer is used to represent recognition categories, i.e. fuzzy rule consequents. Associated with each  $F_2$  category node  $j$  is a vector  $w_j$  of adaptive weights. Initially each weight has the value of one and is said to be uncommitted. After a category is selected for coding it is said to be committed.

Upon presentation of an input vector to the  $ART_a$  module, the bottom-up path of  $ART_a$  computes a complement coding of the input vector. Complement coding is a normalisation rule that preserves amplitude information. For an input vector  $x$  of length  $k$ , the complement coding  $I$  is  $(x_1, x_2, \dots, x_k, 1-x_1, 1-x_2, \dots, 1-x_k)$ . The geometric interpretation of such a coding is a hyperrectangular region with  $x$  in the one corner and  $1-x$  in the corner diagonal to the first corner. From  $I$  the choice function  $T_j$  of each category node  $j$  is then calculated. The output of layer  $F_2$  is determined in a winner-take-all manner. That is, the output of the node  $j$  with the maximum  $T_j$  is set to one. The output of all other nodes are set to zero.





**Figure 2.4 Fuzzy ARTMAP Structure**

In the top-down path of ART<sub>a</sub> the weight vector  $w_j$  of the winner  $F_2$ -node is compared with the input vector  $I$  using a match function. If the match function value meets a so-called vigilance criterion  $\rho$ , i.e.  $r(w_j, I) \geq \rho$ , the network goes into a resonance state resulting in network learning. On the other hand, if the vigilance criterion is not met mismatch reset occurs. In this latter case another  $F_2$ -node is chosen and the match function is recalculated. If no  $F_2$ -node is suitable novelty detection (Carpenter, et al., 1992) takes place whereby an uncommitted  $F_2$ -node is selected. In this instance the vigilance criterion will always be met because the weight vector of an uncommitted node is initialised to the unit vector.

Once the search for an acceptable  $F_2$ -node ends, learning occurs. Only so-called fast learning will be described here. In fast learning the weight vector of the selected  $F_2$ -node is expanded to minimally cover the area defined by the existing weight vector as well as the hyperrectangle formed by  $I$  in attribute space. The Map Field Activation section of ART<sub>b</sub> then maps the selected  $F_2$ -node onto the learned category, or class, of the presented training exemplar. This mapping is learnt by a so-called match-tracking algorithm. In the case of an incorrect mapping the selected  $F_2$ -node is suppressed by increasing the vigilance criterion. This forces the selection of a new,

suitable  $F_2$ -node using the  $ART_a$  module. If no suitable  $F_2$ -node can be found, an uncommitted  $F_2$ -node is selected and linked with the correct mapping category, i.e. output class.

Finally, fuzzy ARTMAP generates fuzzy rules in order to define the similarity between the attribute parts of different training exemplars. This leads to a hyperrectangular discretisation of the attribute space. In other words, the rules are of the 0<sup>th</sup>-order Sugeno type. Each fuzzy rule consists of a conjunction of antecedents. Each antecedent contains a single condition that involves one linguistic variable represented by a single linguistic term. There is one condition for each attribute. No internal disjunction occurs. The vigilance criterion  $\rho$  determines the size of the hyperrectangles that are generated and in turn the size of the membership functions that each represent a linguistic term.

## 2.4 Rule Extraction from Neural Networks

Rule extraction from neural networks involves the conversion of a trained neural network into a more humanly comprehensible form, such as a decision tree or set of “if...then...” rules. This form of rule induction differs from those techniques described in section 2.3.6. In contrast to those methods the neural network is not directly translated into a set of rules. Rather, the rules are extracted using some form of search procedure from the trained neural network.

The TREPAN algorithm (Craven, 1996) is a pedagogical (Andrews, et al., 1995) technique that extracts crisp, decision trees from neural networks. In contrast to earlier techniques (e.g. Fu, 1991; Gallant, 1993; Towell and Shavlik, 1993) TREPAN makes no assumption about the structure of the neural network nor about which method was used to train the network. Furthermore, contrary to earlier techniques (e.g. Seito and Dillon, 1992; Craven and Shavlik, 1994; Tickle, et al., 1994; Thrun, 1995), TREPAN does not assume that the activations of individual hidden nodes act independently and draws no explicit inferences from the magnitudes of connection weights. Craven and Shavlik (1996) report that on three of four test problems considered TREPAN obtains significantly better accuracy than the classification tree algorithms C4.5 (Quinlan, 1993a) and ID2-of-3 (Murphy and Pazzani, 1991). On the fourth test problem TREPAN obtains comparable accuracy. Furthermore, the classification trees extracted by TREPAN are seen by Craven and Shavlik to be as comprehensible as those induced by C4.5 and ID2-of-3.

Another pedagogical technique, the ANN-DT technique (Schmitz, et al., 1999), has been proposed more recently. The technique constructs regression trees from problems with continuous output. Similar to TREPAN, ANN-DT makes no assumptions about the neural

network structure, the network training method, the independence of node activations or the magnitude of connections weights. In contrast to the greedy approach followed by TREPAN, ANN-DT is capable of using a form of look-ahead in the decision tree construction process. Schmitz, et al. (1999) compare the ANN-DT algorithm to CART (Breiman, et al., 1984) on one synthetic problem and two real world regression problems. ANN-DT obtains significantly better regression predictive accuracy to CART on two problems and comparable accuracy on the third problem that was investigated.

Based on the good analytical qualities and reported empirical performance of these algorithms, TREPAN and ANN-DT are taken to be representative of the state of the art in rule extraction from neural networks and will be the only rule extraction algorithms described in detail here.

### 2.4.1 Crisp Rule Extraction Using TREPAN

Like conventional decision tree algorithms, such as C4.5 and CART, TREPAN builds a classification tree by recursively partitioning the input space of the training data. Unlike these algorithms TREPAN uses a neural network, trained on the same data, as a so-called oracle. The aim of the oracle is threefold: to predict the class of an exemplar that is presented to it as a query, to select the best conditions, or splits, for the tree decision nodes and to determine if a tree node covers exemplars of a single class only.

TREPAN recursively splits the original set of training exemplars as the decision tree is generated. As with conventional decision tree algorithms this can detrimentally affect the choice of decision node splits as the depth of the tree increases. In order to minimise this negative effect TREPAN generates query exemplars with the use of the oracle. If the number of original training exemplars falls below some user-defined level, the shortfall is made up with these query exemplars. The oracle generates the query exemplars within the attribute value constraints of the particular decision node for which the query exemplars are required.

The choice of the best decision node  $n$  at which to split next is determined by evaluation of the function  $f(n) = reach(n) \times (1 - fidelity(n))$  for all decision nodes at the bottom of the tree.  $reach(n)$  is the estimated fraction of training and query exemplars that reach node  $n$ .  $fidelity(n)$  is the estimated fidelity of the tree to the trained network for the training and query exemplars. Fidelity is defined as the fraction of training and query exemplars for which both the tree and the network predict the same class. TREPAN splits that decision node that maximises  $f(n)$ , i.e. maximises the fidelity of the decision tree with respect to the trained network.

TREPAN generates  $p$ -of- $n$  expressions as decision node conditions. The  $n$  logical tests may be equality tests for categorical attributes or one-sided interval tests for numerical attributes. When constructing  $p$ -of- $n$  expressions TREPAN enforces a constraint that the same attribute cannot be used in more than one  $p$ -of- $n$  expression in any path between the root node and a given leaf node. This prevents TREPAN having to solve the NP-hard satisfiability problem and to enhance tree comprehensibility. Craven (1996) contends that attributes occurring in multiple  $p$ -of- $n$  expressions may result in complex interactions among antecedent conditions that are difficult to understand.

TREPAN uses a heuristic search to construct the  $p$ -of- $n$  expressions. Selecting the best univariate, binary test for the current decision node initialises the search. The information gain criterion (Quinlan, 1983) is used as an error measure throughout the search to evaluate the quality of candidate split tests<sup>1</sup>. Thereafter TREPAN uses the following two operators to generate candidate  $p$ -of- $n$  expressions based on this initial binary test. The  $p$ -of- $n+1$  operator adds a new value to the set of  $n$  logical tests but holds the threshold,  $p$ , constant. The second operator, denoted  $p+1$ -of- $n+1$ , adds a new value to the set  $n$  of logical tests and increments the threshold  $p$  by one. TREPAN uses a limited form of backtracking to implement truth-preserving modifications to candidate  $p$ -of- $n$  expressions. To prevent overfitting when there is a large amount of training data, and therefore candidate expressions, TREPAN restricts the application of the above-mentioned operators by applying a  $\chi^2$  test after each application of an operator. If the proposed change to the  $p$ -of- $n$  expression significantly changes the existing exemplar partitioning then the proposed change is implemented, otherwise not. Once a  $p$ -of- $n$  expression has been chosen, a postpruning procedure is done to see if any of the logical tests of the  $p$ -of- $n$  expression can be deleted without negatively influencing the error measure. For a given logical test TREPAN first drops the logical test and keeps  $p$  constant and secondly decrements  $p$  by one. TREPAN implements the greatest degree of simplification that is permissible.

TREPAN uses two criteria to halt tree growth, the first a local stopping criterion and the second a global stopping criterion. If a decision node with high probability covers exemplars of a single class only, the local stopping criterion forces the node to be converted to a leaf node. The global stopping criterion is based on a user-defined parameter and limits the size of the tree that TREPAN extracts. Tree size is defined here as the number of decision nodes.

---

<sup>1</sup> A more recent version of TREPAN (Craven and Shavlik, 1996) uses the gain ratio criterion (Quinlan, 1993a) instead of the information gain criterion for improved handling of numerical attributes (Quinlan, 1988).



## 2.4.2 The ANN-DT Rule Extraction Algorithm

In general the tree construction methodology used by ANN-DT is the same as that of TREPAN. A decision tree is generated by recursively partitioning the training data into mutually exclusive subsets. Decision nodes indicate to which subtree the partitioned data belongs. Furthermore, ANN-DT also queries a neural network, that itself has been trained on the entire set of training data, to obtain extra query exemplars. As with TREPAN, the aim of such neural network sampling is to decrease the problems associated with tree construction based on few training exemplars (e.g. suboptimal choice of decision node splits).

In contrast to the  $p$ -of- $n$ -type decision trees derived by TREPAN, ANN-DT extracts univariate, regression trees. Each decision node consists of a univariate, equality test for categorical attributes<sup>2</sup> or a one-sided, interval test for numerical attributes. Each leaf node is described by a constant output value. ANN-DT can use one of two possible methods to select both the attribute and attribute value upon which to split the training data at a new decision node. With the first method, ANN-DT first determines the influence, or significance, of each attribute on the behaviour of the trained neural network model. Refer to Schmitz, et al. (1999) for more details on the significance measure used. The attribute that returns the maximum significance for the neural network model over the given set of data is used to split the current subset of training data. Thereafter, the attribute that minimises the weighted variance of the two subsets of data, generated by splitting the data on that attribute value, is used as the decision node split threshold. The second variant of ANN-DT selects both the attribute and attribute value upon which to next split the training data based on the minimisation of a sum square error criterion.

For either ANN-DT variant recursive splitting of the training data halts when the output variance becomes zero. Premature cessation of tree growth can also occur. After a decision node has been generated, a statistical error measure is used to determine whether the split in the training data is statistically meaningful. If not, the particular decision node is converted to a leaf node and tree growth is halted for this particular sub-branch of the regression tree. This stopping criterion can

---

<sup>2</sup> Schmitz, et al. (1999) state that univariate, one-sided interval tests are used for both categorical and numerical attributes. The reason for this is that  $n$ -valued categorical attributes were presented to the neural network from which ANN-DT extracted regression trees using an  $n$ -valued, orthogonal binary encoding. As described in section 2.3.6.1, the neural network had  $n$  input nodes for each encoded, categorical attribute. Owing to this fact ANN-DT treated categorical attributes in an identical fashion to numerical attributes. However, inspection of the interval tests for a particular categorical attribute show that the tests can be collectively simplified into equality test form.

fail at a given tree depth, even though subsequent splits in the training data may be statistically meaningful. In lieu of this, premature stopping of tree growth while the tree has a depth less than a user-defined minimum is not permitted. A final stopping criterion that is used prevents tree growth from exceeding some user-defined maximum tree depth.

## 2.5 Rule Construction Techniques and the Internet

The rapid growth and improved usability of the Internet has made it possible to access a wide spectrum of sites that provide information regarding rule construction techniques. Owing to the fact that Internet sites sometimes move, have their web addresses changed or are infrequently maintained, only one reasonable stable and well-known web site will be mentioned here.

It is suggested that interested readers who are unfamiliar with the available Internet resources start at the Siftware web page of the kdnuggets web site. The Internet address of this web site is <http://www.kdnuggets.com/siftware.html>. This web page provides a host of pointers to available information and software for, amongst others, decision tree and rule construction techniques. Information regarding public domain as well as commercial software is provided.

In addition, the kdnuggets site publishes a weekly email newsletter (to subscribe see <http://www.kdnuggets.com/subscribe.html>). The information provided in the newsletter pertains to the broad field of data mining and includes notification of important events, a list of recent publications, available tools and services, positions that have become available and courses that are being offered.



# Chapter 3

## Evaluation of Rule Construction Methods

This chapter evaluates and compares the various rule construction techniques that have been described in chapter 2. Rule building techniques that have not been explicitly described in the previous chapter but that are applicable to this discussion are also included in the evaluation.

The first part of this chapter examines the limitations of the search strategies used by the different rule building methods. In particular, the restrictions that are placed on the different strategies in order to make them computationally feasible are discussed and compared with each other. Restrictions that are investigated include greedy search, training data fragmentation and the construction of purely conjunctive rules. Note that this investigation is by no means exhaustive. Rather, attention is focussed on those limitations and drawbacks that the author wishes to address in later stages of this dissertation.

The next part of this chapter concerns the format of the rules that are derived by the various rule construction algorithms. This aspect of rule construction is closely linked with the search strategy that is used to find rules. For example, specific rule formats are chosen so that the search for a set of good quality rules is computationally feasible. In particular, the historical trend in the type of rules that are constructed by the different techniques will be examined in the context of human comprehensibility and intelligibility. It will be shown that the development of the format of “if...then...” rules has primarily concentrated on improving predictive accuracy to the detriment of rule comprehensibility and intelligibility.

The final section gives conclusions on the issues that have been discussed in this chapter. The discussion is supported by a short but more in depth analysis of the results obtained by

techniques that use more sophisticated search than simple greedy search to look for a good set of rules.

The discussion presented in this chapter does not explicitly evaluate rule construction using genetic algorithms. The reason for this is twofold. First, genetic algorithm techniques typically do not suffer from the search strategy limitations discussed in section 3.1. Second, the in depth evaluation of such rule construction techniques, with regard to the search strategy used as well as the rules that are derived, falls beyond the scope of this dissertation. Techniques that use genetic algorithms will therefore not be studied in detail here. However, it should be noted that in chapter 4 reasons are given why genetic algorithms are not used in the fuzzy rule construction technique that is proposed in this dissertation.

### **3.1 Limitations of Search Strategies that Build “if...then...” Rules**

Most of the rule construction techniques described in chapter two use one or more heuristic, search space restriction or both of these to guide the search for a set of good quality rules. Reeves (1993) defines a heuristic as a technique that seeks good (i.e. near-optimal) solutions at a reasonable computational cost without being able to guarantee either feasibility or optimality. In many cases the technique may also be unable to state how close to optimality a particular feasible solution is. This definition will be used throughout this dissertation.

Section 3.1.1 considers the reasons for the use of heuristics and search space restrictions in rule construction algorithms. Thereafter, the limitations of two general rule-building methodologies are examined. The first methodology is discussed in section 3.1.2 and concentrates on decision tree induction and the consequent conversion of the tree to a set of rules. Although tree induction itself does not directly result in a set of “if...then...” rules, it will be examined because a number of algorithms use it as a primary step in the rule-building process. The limitations of decision tree induction therefore indirectly affect the quality of the “if...then...” rules that are derived from the tree.

Section 3.1.3 discusses the limitations and drawbacks of the set covering methodology of rule construction. Crisp rule covering, fuzzy clustering, fuzzy inference networks and radial basis function networks are all grouped under this form of rule construction. The limitations of the remaining fuzzy modelling technique, i.e. genetic algorithms, are in the author’s opinion less detrimental in comparison to those of either the decision tree or set covering approach. The limitations of the genetic algorithm technique will therefore not be explicitly studied here.

### 3.1.1 Computational Complexity of Rule Construction

The first issue that must be resolved is the reason why heuristics and search restrictions are used to construct rules instead of traditional search or combinatorial optimisation techniques (e.g. the Simplex method (Papadimitriou and Steiglitz, 1982)). Consider first the construction of a decision tree. Several aspects of optimal decision tree induction are known to be computationally intractable (Garey and Johnson, 1979). For example, Hyafil and Rivest (1976) as well as Naumov (1991) proved that optimal univariate decision tree construction from decision tables is NP-complete. In other words it is currently unknown whether a deterministic algorithm can solve the tree construction problem in polynomial time. Similarly, Goodrich, et al. (1995) proved for multivariate decision trees that optimal (i.e. smallest) linear decision tree construction is NP-complete even in as little as three input dimensions.

Next consider algorithms that follow the covering approach (e.g. RISE (Domingos, 1994, 1995, 1996a, 1996b, 1996d, 1996e), BEXA (Theron, 1994)). These algorithms usually define rule construction as a set covering problem. A set of rules is induced each of which is said to cover a subset of the training exemplars. Karp (1972) as well as Garey and Johnson (1979) show that the set covering problem belongs to the NP-complete class of problems. Goldschmidt, et al. (1993) state that even when each set covers at most three elements, the set covering problem is already NP-hard.

Techniques that build fuzzy rules directly from data, i.e. not by using an intermediate decision tree (e.g. Yuan and Shaw (1995)), typically use one of three approaches to find the best set of fuzzy rules. The first approach is incremental, i.e. fuzzy rules are added to the existing set of rules in sequential fashion (e.g. the methods of Cho and Wang (1996), Karayiannis and Mi (1997) and Juang and Lin (1998)). The second approach is nonincremental (e.g. the methods of Chen and Xi (1998), and Russo (1998)) and optimises a fixed, user-defined number of fuzzy rules. Both the incremental and nonincremental approaches can be seen as modelling fuzzy rule construction as a set covering problem. Fuzzy rules are said to each cover a fuzzy subset of training exemplars. As stated in the previous paragraph, set covering belongs to the NP-complete set of problems.

The third approach that is used to construct a set of fuzzy rules uses some form of subset selection for either rule structure identification or to select the best subset of rules from a given fuzzy rule set. An example of this approach is the radial basis function network technique of Billings and Hong (1998). Although this method does not directly generate fuzzy rules it adequately illustrates the computational problems associated with subset selection.

Forward orthogonal least squares is used to both select important attributes (and thus indirectly determine the fuzzy rule antecedent part structure) and to select important radial basis functions from the entire set of radial basis functions of the trained network. Owing to the equivalence between radial basis function networks and fuzzy rules, the subset selection problem can be restated as the selection of a subset of fuzzy rules from the total set of fuzzy rules represented by the radial basis function network.

Subset selection in the fuzzy rule context is closely related to subset selection in traditional regression. For example, a set of 0<sup>th</sup>-order Sugeno rules that uses weighted sum defuzzification for output determination can be seen as a regression function that consists of a linear combination of nonlinear terms (the fuzzy rules). Finding the best subset of fuzzy rules is consequently similar to finding the best subset of independent variables (attributes) in the regression problem. The following computational complexity results from the regression domain are therefore applicable to the fuzzy rule subset selection problem.

In univariate regression the selection of the best subset of independent variables using exhaustive search can be computationally exorbitant, especially for large problems (Furnival and Wilson, 1974; Lawson and Hanson, 1995, p.195; Miller, 1990, p.63). For example, the leaps and bounds algorithm (Furnival and Wilson, 1974) for best subset selection through exhaustive search has an operation count of  $6 \times 2^k - 0.5 \times k(k+7) - 6$  where  $k$  is the number of independent variables. For more than about 15 independent variables the required operation count increases exponentially. Miller (1990, p.82) further illustrates the computational expense associated with exhaustive search. Forward selection, backward elimination and exhaustive search were each applied to a regression problem with 25 independent variables and 72 exemplars. It was found that exhaustive search took almost five hundred times longer to complete than forward selection did and roughly ninety times longer than backward elimination did.

In summary, the various ways of constructing either crisp or fuzzy rules typically result in NP-complete, NP-hard or computationally expensive search or combinatorial problems. Heuristics and search space restrictions are therefore necessary to make the various rule construction techniques computationally feasible. The following sections examine these heuristics and restrictions in more detail to determine some their overall limitations and drawbacks.

### 3.1.2 Limitations of Rule Building by the Decision Tree Route

Two limitations of decision tree induction will be discussed here. These are the suboptimality of greedy tree induction and the detrimental effect of training set fragmentation and concept replication on predictive performance. Thereafter the primary limitations of existing techniques that convert decision trees to sets of “if...then...” rules are discussed.

#### 3.1.2.1 Optimality of Decision Tree Induction

Owing to the computational complexity of decision tree induction, decision tree algorithms typically follow a greedy approach when constructing a decision tree. Such an approach reduces the computational complexity to a level that increases approximately linearly with increasing problem complexity (Quinlan, 1983; Saravia and Stephanopoulos, 1992). In short, a tree is induced in a top-down, depth-first fashion with locally optimal choices made at each decision node without lookahead or backtracking. As described in chapter two, this is the approach followed by C4.5 (Quinlan, 1993a), CART (Breiman, et al., 1984) as well as the fuzzy decision tree technique of Yuan and Shaw (1995). The TREPAN algorithm (Craven, 1996) also performs greedy tree induction but expands the tree in a best-first manner rather than depth-first manner. Furthermore, TREPAN employs a limited form of backtracking at decision node level to ensure truth-preservation when deriving a  $p$ -of- $n$  test for a specific decision node.

Although efficient, greedy tree induction may get trapped in local optima and thus degrade the quality of the induced tree (Rendell and Seshu, 1990). Techniques have therefore been devised that predominantly use so-called lookahead to improve on greedy tree induction. Owing to its relatively extensive use lookahead will be examined exclusively here.

Norton (1989) studied lookahead using a variant of ID3 (Quinlan, 1986) called IDX. A single data set was investigated. Lookahead was found to reduce the average depth of the induced decision trees. Ragavan and Rendell (1993) compared ID3 with a variant that uses lookahead. Lookahead was constrained by both geometric constraints and by limiting the depth and breadth of search. On four synthetic, Boolean data sets the lookahead variant produced decision trees that were significantly more accurate than those obtained greedily. Wallace and Patrick (1993) induced decision trees based on the Minimum Description Length (Rissanen, 1983) principle. Six different problems were studied. Up to four-level lookahead was performed on categorical attributes. No lookahead was done for numerical attributes owing to the computational unfeasibility of performing exhaustive searches on the numerical attributes

to determine split values. Wallace and Patrick found that in general lookahead produced slightly better predictive performance in comparison to a greedy approach. Furthermore, the size of the trees derived using lookahead was found to be significantly smaller than that of greedily induced trees.

Murphy and Pazzani (1994) conducted extensive tests with four classification data sets. All consistent decision trees were generated for these data sets and the resulting trees compared to those obtained by greedy induction. In contrast to the above results, it was found that exhaustive search for the simplest (i.e. smallest) consistent trees did not necessarily lead to improved predictive performance. More recently, Murthy and Salzberg (1995) as well as Murthy (1996) extensively studied classification tree induction with one-level lookahead on a number of synthetic and real-world data sets. (Both publications discuss exactly the same investigation.) Murthy and Salzberg found that performing one-level lookahead instead of greedy induction generally produced slightly shallower trees but with approximately the same classification accuracy and overall tree size. In addition, in several situations limited lookahead produced worse (less accurate, larger) trees.

From the above results it is difficult to determine the overall efficacy of lookahead (as one possible improvement over simple greedy induction) in decision tree induction. Furthermore it is not always clear whether improved results are directly attributable to lookahead, or to some other algorithmic improvement. Despite these inconsistencies the following observations can be made at this stage. Limited lookahead does not guarantee improved predictive performance. On the other hand, algorithms that use lookahead with a search depth of more than one and that perform such lookahead intelligently (e.g. with the geometric constraints used by the algorithm of Ragavan and Rendell (1993)) can construct trees that exhibit improved predictive performance in comparison to greedily-induced trees.

One final observation that can be made with certainty is that the computational complexity of performing exhaustive search or even search with a predefined beam width severely restricts and limits the depth to which lookahead in decision trees can be performed. This is especially true with numerical attributes where an exhaustive search must typically be performed at each decision node to find the best numerical split value (see e.g. Wallace and Patrick (1993)). This in turn has made the use of lookahead with the aim of producing better decision trees or even to explicitly determine the effects of extensive lookahead itself, computationally prohibitive.

As an aside, the lookahead experimentation by Murthy (1996) is fairly extensive and was therefore examined in more detail than is explained here. It is argued that the generally poor



results obtained by Murthy are not entirely because tree induction using lookahead, as means of improving on greedy induction, is incapable of constructing better decision trees. It is contended that one of the primary reasons for the poor results is because of the poor quality of the data that were investigated. This phenomenon of using poor data is not restricted to tree induction with lookahead only and therefore further discussion is postponed until the limitations of other rule construction techniques are considered in more detail. Specifically, this issue is discussed in section 3.3.

### 3.1.2.2 Training Set Fragmentation and Concept Replication

Greedy decision tree induction suffers from two further problems that can degrade both the predictive performance and the comprehensibility of “if...then...” rules that are derived from such trees. These are the concept replication problem and the training set fragmentation problem (Pagallo and Haussler, 1990). The replication problem forces duplication of subtrees in crisp disjunctive concepts such as “if ( $A$  and  $B$ ) or ( $C$  and  $D$ )”. In other words, individual subconcepts in the data may become fragmented. The fragmentation problem causes the partitioning of the training data into fragments. Both these two problems reduce the number of exemplars at lower nodes in the tree. These exemplars are needed for statistical significance of tests performed in the lower levels of tree during the tree construction process (Mingers, 1987). The net result of these problems is that suboptimally performing and overly complex trees are usually induced (Breiman, et al., 1984; Quinlan, 1993a; Kim and Koehler, 1996). This problem is exacerbated if the training exemplars contain significant noise. Tree pruning (Mingers, 1989; Esposito, et al., 1995) is typically employed to minimise these effects.

Of concern here is the effect of the replication and fragmentation problems on decision tree overall predictive performance, even after tree pruning has been applied. Although the TREPAN technique (described in section 2.4.1) is designed primarily for rule extraction from neural networks, comparative results published for this technique aptly illustrate the degradation effect of the replication and fragmentation problems. (No other meaningful comparative results could be found in the literature available to the author.) Craven and Shavlik (1996) compare the ID2-of-3 algorithm (Murphy and Pazzani, 1991) to the TREPAN algorithm on four classification problems. The ID2-of-3 algorithm and TREPAN both greedily induce decision trees with  $p$ -of- $n$  splits. In contrast to ID2-of-3, TREPAN generates extra query exemplars in order to minimise the negative effect of few training exemplars lower down in the decision tree (see section 2.4.1 for more details on the specific methodology used). Note that these query exemplars are based on and labelled by the trained

neural network model of the oracle and as such are not bona fide training exemplars. On all four problems TREPAN obtains better predictive performance than ID2-of-3. In addition, the average size of the trees induced by TREPAN is significantly smaller in comparison to that of ID2-of-3. These results show that the fragmentation of training data caused by simple greedy induction (without using something like an oracle to generate query exemplars) can therefore detrimentally affect both predictive accuracy and rule simplicity, even if tree pruning is applied.

### 3.1.2.3 From Trees to Rules

Large decision trees are often difficult to understand because each logical test in the tree has a specific context that is established by the outcomes of tests at parent decision nodes (Quinlan, 1993a). This unique context is crucial to the correct understanding of the test. It is often difficult to keep track of the continually changing context while examining a large tree. In addition, as described above, individual subconcepts in the data may either be replicated or become fragmented. Conversion of a decision tree into set of crisp rules is designed to minimise these effects and thus improve the comprehensibility of the trained tree model.

C4.5Rules (Quinlan, 1993a) and the technique of Yuan and Shaw (1995) convert induced decision trees to “if...then...” rules. Inspection of these algorithms (see sections 2.2.1.4 and 2.3.4.2, respectively) show that both remove conditions from rules in a greedy fashion<sup>1</sup>. One of the reasons for this approach is that selecting the best subset of conditions from the antecedent part of a particular rule is again a subset selection problem. As described in section 3.1.1, the subset selection is computationally expensive, especially for large problems.

In the context of decision trees, Mingers (1987) found that attributes can work together to realise statistically significant tree branches in lower levels of the induced tree. This can occur even if the particular branch is not statistically meaningful higher up in the tree. Kononenko, et al. (1997) found that the Lookahead Feature Construction (LFC) algorithm is able to successfully solve problems that have attributes with strong conditional interdependencies. For example, the LFC algorithm was able to successfully solve a parity problem using lookahead (in feature construction) that a greedy LFC algorithmic variant could not. Finally,

---

<sup>1</sup> Note that in terms of finding the best subset of rules (and not the conditions for a particular rule) C4.5 does perform subset selection. For small rule sets exhaustive subset selection is performed. A form of simulated annealing is used to find a good subset of rules for large rule sets (Quinlan (1993a), p. 53).

Greene and Smith (1993) state that greedy tree induction can struggle to find an optimal solution in complex problem domains that exhibit significant attribute interaction. They show that genetic algorithm-based construction of rules, which is able to discover such interaction, gives superior classification results to those obtained by the CN2 covering algorithm (Clark, 1989) and to NewID (a variant of ID3).

Apart from the general problem of getting stuck in local optima and thus producing a suboptimal set of simplified rules, the greedy simplification methods of Quinlan (1993a) and Yuan and Shaw (1995) are unable to explicitly discover and exploit these conditional dependencies between attributes. The net result of this latter limitation is that acceptable rule simplifications that involve more than one interdependent antecedent will not be discovered nor implemented. In other words, greedy simplification will not always produce the most compact and comprehensible set of rules while maintaining acceptable predictive accuracy. (Remember the aim of C4.5Rules and the technique of Yuan and Shaw is rule simplification for improved comprehensibility and not necessarily for improved classification accuracy.)

### **3.1.3 Limitations of Rule Building by the Set Covering Approach**

This section considers the limitations and drawbacks of those techniques that build rules using the set covering approach. These techniques include both crisp rule covering methods (e.g. RISE) and fuzzy modelling techniques (e.g. the ellipsoidal classifier, SONFIN). In particular, this section concentrates on three shortcomings of the various techniques. The reason for this is that although there are more problems associated with the various techniques, this dissertation seeks to primarily address these specific deficiencies only. The first two limitations are usually found in crisp rule covering algorithms but not in fuzzy methods. These are training set fragmentation and greedy search of the best set of rules. In contrast, the third limitation is prevalent in fuzzy set covering techniques. This limitation is the inability to build internally disjunctive fuzzy rules.

#### **3.1.3.1 Problems Regarding Training Set Fragmentation**

Covering algorithms that build crisp, classification rules typically construct rules in a “separate and conquer” fashion. AQ15 (Michalski, et al., 1986), BEXA (described in section 2.2.2.2), CN2 (Clark, 1989), RIPPER<sub>k</sub> (Cohen, 1995), RULES-3 Plus (Pham and Dimov, 1997) and Swap-1 (Weiss and Indurkha, 1993) are all algorithms that use this approach. In short, one rule is induced at a time. Once construction of the rule is halted the rule is stored

and those data that it covers are removed from the training data set. The remaining data are then used to construct another rule, and so on.

Such rule construction can suffer in similar fashion from the data fragmentation problems described for tree induction in section 3.1.2.2. In particular, such greedy covering may degrade the predictive performance of the generated rules. This is especially true for those rules induced in the latter stages of rule construction. In this case the covering algorithm will have increasingly little data upon which to base rule specialisation, data needed for statistical significance of any specialisation that is implemented. This aggravates the small disjuncts problem (Holte et al., 1989): rules covering few exemplars (less than five or six) tend to be highly error-prone, but removing them often increases the global error even further. This problem is exacerbated by the presence of noise in the data. Both prepruning (premature termination of rule construction) and postpruning (removal of statistically insignificant antecedents and rules) are used to minimise these negative effects on predictive performance.

As with tree induction, such pruning mechanisms do not always entirely make up for the degradation in predictive performance. Consider the experimental results obtained by the RISE algorithm (section 2.2.2.1) and the R-MINI rule-generation algorithm (Hong, 1997) in comparison to the benchmark “separate and conquer” algorithm CN2. RISE and R-MINI were chosen because both algorithms, in contrast to “separate and conquer” algorithms, use all the training data during the construction of the entire set of rules. Both algorithms start with each exemplar being defined as a rule. Both contain a generalisation step to generalise rules. R-MINI contains a further specialisation step to make rules minimally specific. The particular results reported for RISE are found in Domingos (1996e) and for R-MINI in Hong (1997).

Domingos compares RISE to CN2 on thirty data sets. Based on these thirty results, RISE is found to be significantly more accurate (at a 99.6% confidence level) than CN2 according to the Wilcoxon signed-ranks test (DeGroot, 1986). R-MINI is compared to CN2 on eight data sets. R-MINI is found to be more accurate than CN2 on seven of the eight data sets. No significance tests are reported for these results. In summary, these results illustrate that training data fragmentation can have a significantly detrimental effect on the overall accuracy of rules that are generated by “separate and conquer” rule covering algorithms.

### 3.1.3.2 Shortcomings of Greedy Search Methods

As with greedy tree induction, greedy rule covering can get stuck in local optima and thus produce a suboptimal set of rules. By greedy rule covering is primarily meant that each rule is

generated on its own. Existing rules are not updated in any fashion if a new rule is created. Examples of set covering algorithms that follow the greedy approach are AQ15, CN2, BEXA, RISE, Rules-3 Plus, RIPPER $k$  and Swap-1.

As with tree induction using lookahead (section 3.1.2.1), the results obtained for crisp rule covering methods that use more sophisticated search methods do not unequivocally show that improved search methods generate improved predictive performance (Quinlan and Cameron-Jones, 1995). Quinlan and Cameron-Jones contend that the reason for such ambiguous results is that sophisticated search methods can “oversearch”. In other words, such search methods can find models that fit the training data well (better than models found through greedy search) but exhibit poor predictive performance. Greedy methods are less likely to find such models because they search a much smaller model space.

As a partial remedy, Quinlan and Cameron-Jones propose a “layered search” method that commences with greedy search and then extends the search scope for a number of iterations until a stopping criterion is met. Increasing the search beam width extends the search scope. As this technique is seen to be the state-of-the-art in search methods for set covering algorithms that are less greedy than simple greedy search, the results obtained for this algorithm will be examined in more detail.

Quinlan and Cameron-Jones compare the predictive accuracy<sup>2</sup> and complexity of the rules generated by the layered search method on twelve, real world data sets to those of a greedy rule-covering algorithm. An extensive search method that approximates exhaustive search is also included in the comparison. It is reported that layered search is significantly (at a 99.5% confidence level) more accurate than greedy search in five problems and significantly less accurate in three problems. In comparison to the extensive search method, layered search is significantly more accurate in six problems and worse in one.

---

<sup>2</sup> The comparative accuracy significance results obtained by Quinlan and Cameron-Jones (1995) should be treated with caution. For each problem that was considered, the data were randomly divided 500 times into equal-sized training and testing sets. Thereafter a paired, two-tailed t-test was used to test the significance of the results obtained by the layered search method in comparison to the other algorithms that were considered. Salzberg (1997) states that there are significant problems associated with this experimental design. For example, the t-test assumes that each of the 500 trials is independent of each other. This is not the case in the experiments of Quinlan and Cameron-Jones. It is difficult to determine the magnitude of the effect of not performing independent experiments. The reported significance results should therefore be evaluated with care.

Of interest also is the comparative complexity of the rules derived by the various methods. Consider first the number of rules that was generated. Layered search generated on average 33.0% fewer rules in comparison to greedy search. Extensive search constructed on average 42.4% fewer rules than greedy search. Second, consider the total number of antecedents of each of the generated rule sets. Layered search derived rules with on average 17.4% fewer antecedents in comparison to greedy search. Extensive search generated rules with on average 19.7% fewer antecedents than greedy search. (No significance tests were reported.)

In summary, these results show that layered search, as an example of an improvement over simple greedy search in rule covering algorithms, can find less complex models with improved predictive accuracy.

Note that fuzzy rule construction methods that generally follow the set covering approach typically do not use greedy learning as a primary means of obtaining rules. By greedy learning is meant that a new rule is generated without changing previously constructed rules. Rather, if such an incremental approach is used, both the new rule and the existing set of rules will usually be continuously optimised (e.g. SONFIN, the GRBF network). If the fuzzy rule construction method does contain a greedy component, it is typically used in some secondary component of rule construction. For example, the GRBF network (section 2.3.6.5) can use greedy recursive least squares to optimise the weights to the output layer of the radial basis function network.

An exception is the nonincremental method of Billings and Hong (1998). This method uses greedy orthogonal least squares to select important attributes and important radial basis functions. The reason for this is that both attribute and radial basis function selection are subset selection problems (see 3.1.1 for details on the problems associated with subset selection).

Unfortunately, as discussed in section 2.3.1, few fuzzy modelling publications compare their proposed fuzzy rule construction technique to other techniques on a significant number of problems. Furthermore, in practically all cases (bar the ubiquitous Iris data set) fuzzy methods are evaluated on different data sets to those typically used to evaluate crisp rule construction techniques. It is therefore not possible to meaningfully compare the relatively nongreedy search methods used in fuzzy modelling with the greedy approaches used in crisp rule covering.



### 3.1.3.3 Restriction to Building Conjunctive Rules Only

The final limitation of set covering algorithms that will be discussed is the inability of many crisp rule covering methods and most, if not all, fuzzy modelling techniques (that follow the set covering approach) to build internally disjunctive “if...then...” rules. Examples of crisp methods that are limited to purely conjunctive rules only are CN2, RISE, R-MINI, RULES-3 Plus and BRUTE (Riddle, et al., 1994). PVM (Weiss, et al., 1990), AQ15 and BEXA are examples of crisp methods that are able to build internally disjunctive rules.

In terms of fuzzy modelling, the clustering methods described in section 2.3.3 build purely conjunctive rules only. As described in section 2.3.6.4 through 2.3.6.6, SONFIN, the GRBF network and fuzzy ARTMAP can also only build rules with purely conjunctive antecedent parts. Inspection of the other neural network methods listed in section 2.3.6.3 and that use some form of set covering approach to build fuzzy rules, shows that these techniques can also only build purely conjunctive rules. The specific methods that were examined are those proposed by Simpson (1992, 1993), Sun (1994), Nie (1995), Cho and Wang (1996), Lotfi and Tsoi (1996), Fritzke (1997), Hwang and Bang (1997), Langari, et al. (1997), Wang, et al. (1997), Yingwei, et al. (1997) and Cai and Kwan (1998).

Since so few algorithms generate internally disjunctive rules, the question might reasonably be asked, does the ability to construct internally disjunctive rules make a difference? Unfortunately, few comparative evaluations have been published. One comparison that has been made is between the two crisp set covering algorithms, BEXA and CN2 (Theron and Cloete, 1996). As mentioned before, the former can build internally disjunctive rules whereas the latter can generate purely conjunctive rules only. Ten different classification problems were investigated of which one was specifically designed to determine the advantage of using internally disjunctive instead of purely conjunctive rules.

In terms of predictive accuracy, Theron and Cloete report that the rules generated by BEXA are significantly more accurate than those of CN2 on two of the ten problems considered. One of these is the internal disjunction problem. The confidence levels are 97% and more than 99.99%, respectively. The rules of BEXA are less accurate (98% confidence level) than those of CN2 on a single problem. There is no significant difference between the accuracy results obtained for the other seven data sets.

In addition, Theron and Cloete, as a measure of rule model complexity, compared the total number of antecedents of the respective rule sets generated by BEXA and CN2. The rules

derived by BEXA were significantly simpler (with greater than 99.99% confidence) to those of CN2 for nine of the ten problems considered. This included the internal disjunction data set where BEXA produced a rule set with 97.7% fewer antecedents in comparison to the rule set obtained by CN2. Theron and Cloete state that the overall improved performance is as a result of the richer description language (BEXA can do internal disjunction as well as negation), optional search restrictions (as described in section 2.2.2.2), a stop-growth test as well as the capability of BEXA of performing postpruning of rules.

From these results it is clear that rule construction techniques that are able to build internally disjunctive rules and that can intelligently restrict rule specialisation, are able to construct rules that can be more accurate and significantly simpler in comparison to those generated by algorithms that do not possess this functionality. This observation is further substantiated by empirical results from the research area of rule extraction from neural networks. Craven (1996) studied four rule extraction techniques (Saito and Nakano, 1988; Towell and Shavlik, 1993; Craven and Shavlik, 1994; Thrun, 1995) that generate purely conjunctive rules only. It was found that these techniques often do not scale well to difficult problems. This in turn means the rule models that are generated for such problems are often large and difficult to understand.

## 3.2 Rule Comprehensibility

The concept of comprehensibility as it is used in this section is based upon the discussion given in sections 1.1 and 1.2. This includes the comprehensibility postulate of Michalski (1983). To recap, models that use a concept representation that is understandable to specifically chemical process operators, with similar capabilities and education to those described in section 1.1 and 1.2, are seen as the most comprehensible and intelligible. In the context of the comprehensibility postulate this implies models that use only natural language constructs are seen as more comprehensible than those which contain mathematical constructs such as linear equations, etc. The reason for this particular definition or description of comprehensibility is that one of the aims of this dissertation is to develop process models that are understandable and useful to primarily human operators of large chemical processes.

Section 3.2.1 identifies and justifies the research fields that will be investigated. Different rule construction research fields are currently at different levels of maturity and therefore not all fields will be studied. Section 3.2.2 and 3.2.3 examine the historical trend in the types of rules that are developed by rule construction algorithms. The former considers the crisp decision

tree field and the latter the field of fuzzy modelling. It will be shown that a growing number of algorithms improve rule model predictive performance by enhancing the mathematical content of the rules that are generated. For example, 1<sup>st</sup>-order Sugeno rules, each with a consequent comprising of a linear function of attributes, are used instead of 0<sup>th</sup>-order Sugeno rules (where a rule consequent is a numerical constant). The effect of these mathematical enhancements will be discussed in terms of how they effect predictive performance as well as rule comprehensibility and intelligibility.

### 3.2.1 Historical Trends in Rule Format

Although this dissertation focuses on techniques that build “if...then...” rules and not on those that induce decision trees, crisp tree induction will be briefly studied in the first part of this discussion, i.e. section 3.2.2. Note that this description is by no means exhaustive. Refer to Murthy (1996) for a more in depth survey of different decision tree techniques and the types of trees that they induce.

The primary reasons for the inclusion of a summary on the evolution of decision tree format are as follows. First, the field of crisp tree induction is relatively mature. Analysis of trends in crisp tree format will therefore provide a good perspective of how a particular field has altered the format of the models that are generated in order to enhance model performance. In the author’s opinion, the same general trends are being followed in the relatively new fields of rule construction (e.g. fuzzy modelling using radial basis function networks), fuzzy rule modelling in particular. Second, techniques such as C4.5Rules (Quinlan, 1993a) and the technique of Yuan and Shaw (1995) use decision trees as a basis for building rules. Changes to the types of trees that are induced will therefore directly influence such tree to rule conversion techniques (although the author is unaware of a working algorithm that converts more complicated trees than those generated by C4.5 or the technique of Yuan and Shaw to rules). The final reason is that TREPAN, in contrast to most other rule extraction methods, builds a decision tree model and not a set of “if...then...” from a trained neural network. Decision trees are therefore also included in the discussion so that the comprehensibility of trees induced by TREPAN can be placed in context.

The second part of this discussion (section 3.2.3) examines methods that construct rules using fuzzy modelling. In particular, the general fuzzy modelling methodologies described in section 2.3 will be studied. Based upon the publications available to the author, existing nonhybrid methods that induce fuzzy decision trees, or that explicitly generate fuzzy rules

using either genetic algorithms or radial basis function networks do not construct rules that are more complex than the Mamdani or Sugeno types of rules described in section 2.1.2.3. These three research fields will therefore not be studied here. The single exception is the genetic algorithm-based technique of Yuan and Zhuang (1996). In contrast to the purely conjunctive antecedent format of the Mamdani or Sugeno rules, this technique can construct fuzzy rules that possess internal disjunction.

Those methods that generate crisp rules using the set covering approach will also not be explicitly examined here. The reason for this is that, in the author's experience, most set covering methods do not construct rules that are more complex than those induced by BEXA. Such rules do not contain mathematical constructs to aid performance and therefore, in the author's opinion, follow comprehensibility postulate of Michalski (1983), described in section 1.2, fairly closely. An exception is the group of set covering methods that use constructive induction. During rule construction, these methods generate new attributes from existing attributes using either numerical or logical operators. See Arciszewski, et al. (1995), Szczepanik, et al. (1995) and Bloedorn and Michalski (1991, 1996) for more details regarding constructive rule induction. Methods that employ constructive induction will not be examined here owing to the fact that they by nature generate less comprehensible rules than methods that do not use constructive induction (e.g. BEXA).

### **3.2.2 The Changing Appearance of Crisp Decision Trees**

The first decision tree algorithm that will be considered is ID3 (Quinlan, 1979, 1983, 1986). The reason for this is that the comprehensibility of ID3-induced decision trees is taken here as the standard against which the comprehensibility of all other types of trees is measured. In other words, it is assumed that the concept descriptions used by ID3 are understandable to chemical process operators. ID3 generates classification trees with the condition at each decision node a test on a single attribute. This pertains to both categorical and numerical attributes. Each leaf node is assigned a single class label. This tree format is the same as that used by C4.5 (described in section 2.2.1).

#### **3.2.2.1 Enhancements to Tree Split Functions**

Breiman, et al. (1984) proposed the CART set of algorithms (also described in section 2.2.1). In contrast to ID3 and C4.5, CART is able to generate numerical multivariate splits to improve predictive performance on problems with decision surfaces that are not parallel to the attribute space axes. A numerical multivariate split comprises of a Boolean test on an attribute

that is a linear combination of original continuous attributes. Likewise, a categorical multivariate split comprises of a Boolean test on an attribute that is a purely conjunctive combination of original categorical attributes. A number of other researchers have since developed algorithms that are also capable of generating decision trees with linear, multivariate splits on numerical attributes. These use more advanced search techniques than CART to find the best splits and thus improve tree performance (e.g. Heath, et al. (1993), Brown, et al. (1996) and Murthy (1996)).

The univariate tests of the ID3-induced tree format have been extended in a number of other ways as well. For example, Seshu (1989) implemented decision trees that use parity operators in the tests at decision nodes to improve performance on parity problems. These operators are applied to a set of Boolean attributes and return TRUE if an odd number of the attributes have the value one. Instead of univariate splits, the ID2-of-3 algorithm of Murphy and Pazzani (1991) generates so-called Boolean threshold functions, or  $p$ -of- $n$  splits. These are the same splits generated by TREPAN during tree induction. Brodley (1995) and Ting (1994) designed algorithms that enhance tree performance by incorporating other classifiers (e.g.  $k$ -means clustering) into the basic ID3-type tree format as decision nodes. Ragavan and Rendell (1993) as well as Kononenko, et al. (1997) use lookahead feature construction to build new attributes using logical operators such as conjunction, disjunction and negation for improved accuracy in domains with strong conditional interdependencies between attributes. Zheng (1995) studied  $x$ -of- $n$  tests. In contrast to the above-mentioned  $p$ -of- $n$  tests, the  $x$ -of- $n$  test returns an integer value rather than a Boolean output. As in  $p$ -of- $n$  tests (described in detail in section 2.1.1)  $n$  represents the cardinality of a set of attribute-value pairs but the value  $x$  is the number of these pairs that are true.

### 3.2.2.2 Enhancements to Leaf Node Representations

The original ID3 and C4.5 leaf node representation, i.e. a class label, has also been changed to improve tree performance on both classification and regression problems. For example, Utgoff (1989) proposed perceptron trees. These trees have univariate splits and linear threshold units as leaves. Utgoff and Brodley (1990) updated this tree format to include multivariate splits at all decision nodes. The M5 algorithm (Quinlan, 1992) generates an updated ID3 tree format that can handle continuous classes. Each leaf node is assigned a linear function of relevant attributes instead of a class label. The M5' algorithm (Wang and Witten, 1997; Frank, et al., 1998) derives a similar type of tree. In contrast to normal methods of predicting the output of an unseen exemplar used in the ID3-style tree format, both M5 and



M5' use the parent nodes of a leaf node to mathematically smooth the initial output prediction of the leaf node function, given an unseen exemplar. Torgo (1997) proposed binary, kernel regression trees that use nonlinear Gaussian kernel functions as leaves.

Finally, the predictive performance of decision trees has been enhanced with the use of bagging and boosting (Breiman, 1996; Quinlan, 1996) as well as committees of decision trees (Heath, et al., 1996). Both form a set of decision trees that are combined by some form of voting scheme. In other words, to determine the output of an unseen exemplar each decision tree of the ensemble under consideration is used to predict the output. The voting scheme is then used to determine the final predicted output, based on the results obtained from the individual decision trees.

### 3.2.2.3 Improvements in Tree Accuracy Owing to Tree Format Enhancements

The obvious question now is, does the improvement in decision tree predictive accuracy justify the decrease in tree comprehensibility owing to these decision tree format adaptations? Although comparative results for the algorithms mentioned above are few, consider the following summary given in table 3.1 of results available to the author. Note that only the average predictive accuracy of the various tree models is compared<sup>3</sup>. Some publications report accuracy performance for a number of advanced decision tree algorithmic variants. If this is the case, the best accuracy of any variant for a particular data set was used to determine the average performance results presented in the table. Finally, where applicable, the performance of the  $k$ -DT committee of classification trees was compared to that of the most accurate and smallest SADT tree in the  $k$ -DT committee.

OC1 generates classification trees with numerical multivariate splits. ID2-of-3+ is an enhanced version of ID2-of-3 proposed by Craven (1996). MCS stands for the Model Class Selection system (Brodley, 1995). The technique automatically selects the type of split function (e.g. univariate split,  $k$ -means clustering, etc.) at each decision node in the tree. LFC stands for the Lookahead Feature Construction classification tree algorithm (Kononenko, 1997). Both the MCS and LFC algorithms are compared to univariate classification trees

---

<sup>3</sup> The comprehensibility of different types of tree models is most often either ignored or quantified in terms of relatively poor descriptors such as average tree size. A descriptor such as tree size does not quantify the difference in comprehensibility between, for example, a univariate splitting test and a multivariate splitting test. Smaller trees therefore do not automatically indicate more comprehensible tree models. Such indicators of tree model comprehensibility will therefore not be compared with each other.



induced in a greedy, “divide and conquer” recursive fashion similar to the method used by C4.5. M5’ is compared to an updated version of C4.5 called C5.0. Quinlan (1996) compares the original C4.5 to variants that are either bagged or boosted to improve predictive performance. Finally, Heath, et al. (1996) compare a committee of classification trees collectively denoted as  $k$ -DT to a single SADT tree. The SADT algorithm induces multivariate classification trees. The multivariate splits are determined with the aid of simulated annealing (Kirkpatrick, et al., 1983).

Advanced Decision Tree Algorithm	Benchmark Algorithm	Reference	Number of Data Sets Considered	Average Improvement in Classification Accuracy (%)
OC1	C4.5	Murthy (1996)	6	0.8
ID2-of-3+	C4.5	Craven (1996)	6	1.0
MCS	univariate tree	Brodley (1995)	16	2.9
LFC	univariate tree	Kononenko (1997)	13	-1.2
Perceptron Tree	C4.5	Brodley, et al. (1995)	8	2.7
M5’	C5.0	Frank, et al. (1998)	33	1.4
Bagged C4.5	C4.5	Quinlan (1996)	27	1.6
Boosted C4.5	C4.5	Quinlan (1996)	27	2.3
$k$ -DT	SADT	Heath, et al. (1996)	3	-0.1%

**Table 3.1 Average Improvement in Predictive Accuracy of Advanced Decision Tree Algorithms over Univariate Decision Tree Algorithms**

Inspection of table 3.1 indicates that the predictive accuracy of decision trees that use an enhanced format (e.g. multivariate splits, linear models as leaves) are on average 1.3% more accurate than univariate trees derived by C4.5, etc. In the author’s opinion this increase in accuracy in general does not justify the degradation in tree comprehensibility even though in most cases the performance improvement is statistically significant<sup>4</sup>. In most cases increasingly complicated mathematical or statistical constructs have been incorporated into the decision tree, rather than more humanly comprehensible natural language representations. It is true that such knowledge compaction has often led to smaller, supposedly more intelligible trees but even so it is becoming increasingly difficult to understand decision trees, especially for those with little or no mathematical or statistical background. This is in direct

---

<sup>4</sup> In defence of such advanced decision tree algorithms it must be stated that for specific types of problems such algorithms can perform substantially better than simple univariate decision tree algorithms. For instance, Kononenko (1997) reports that the LFC algorithm that uses both feature construction and lookahead search is able to solve a parity problem that a greedy univariate tree algorithm cannot.

contrast to one of the original aims of generating a decision tree, i.e. capturing knowledge in the form of an easily understandable and comprehensible tree model.

### 3.2.3 The Changing Appearance of Fuzzy Rules

The field of fuzzy modelling is relatively young in comparison to that of crisp tree induction. Even so, the following discussion will show that, in a similar fashion to the trend in tree induction, fuzzy rules are becoming increasingly more complicated and therefore more difficult to understand. The first part of this discussion illustrates the early evolution of fuzzy rules from purely fuzzy rules to rules containing increasing amounts of mathematical constructs. Thereafter the relevant fuzzy modelling methods are studied to determine how these earlier fuzzy rule formats have been further enhanced to improve predictive performance.

One of the first fuzzy rule formats that was proposed in fuzzy modelling was the Mamdani format (Mamdani, 1975). As described in section 2.1.2.3, both the antecedent part and consequent of a Mamdani fuzzy rule is described using natural language concepts such as linguistic variables. Defuzzification is typically used to determine the crisp output of a set of Mamdani fuzzy rules, given some crisp input. Thereafter the Tsukamoto fuzzy rule format (Tsukamoto, 1979) was proposed. Instead of the fuzzy output produced by a Mamdani fuzzy rule, a Tsukamoto fuzzy rule infers a crisp, numerical output. Instead of using defuzzification as in a Mamdani rule system, the overall output of a Tsukamoto rule system is obtained by taking the weighted average of each rule's output. After the development of the Tsukamoto fuzzy rule model, the Sugeno fuzzy model (Takagi and Sugeno, 1985; Sugeno and Kang, 1988) was proposed. As described in section 2.1.2.3, the fuzzy rule consequent was replaced by a linear function of the original crisp problem attributes.

Next, consider methods that specifically build fuzzy rules using the clustering approach. Inspection of the publications available to the author shows that most clustering methods (e.g. Wang and Langari (1995, 1996b), Chen and Xi (1998)) build higher-order Sugeno rule models. The membership functions representing the linguistic terms in the antecedent part of each rule are axis-parallel and therefore functions of single linguistic variables only.

An exception is the recently proposed technique of Kim, et al. (1997). This technique derives oriented, hyperellipsoidal clusters in the attribute space. In other words, in contrast to the relatively intelligible membership functions generated by the methods of Wang and Langari as well as Chen and Xi, these membership functions represent linguistic terms that are based

on linear combinations of the original linguistic variables. For example, a typical rule might be “if  $f$ (temperature, pressure) is *High* and  $g$ (temperature, pressure) is *Generally Low* then the reaction rate is  $h$ ”.  $f$  and  $g$  represent Gaussian membership functions of combinations of the original attributes.  $h$  is a linear function of the original attributes. “*High*” and “*Generally Low*” are linguistic terms based on combinations of the original linguistic variables. It should be clear that such a rule is more difficult to understand than normal Sugeno rules.

Similar adaptations are made by SONFIN (described in section 2.3.6.4). In contrast to the standard Mamdani or Sugeno rules derived by most earlier fuzzy inference network approaches (e.g. Jang, 1993; Sun, 1994; Lotfi and Tsoi, 1996; Wang, et al, 1997; Cai and Kwan, 1998), the advanced SONFIN constructs  $n^{\text{th}}$ -order Sugeno fuzzy rules, each with an antecedent part formed by an oriented hyperellipsoid. As described above, the membership functions derived from such hyperellipsoids are functions of two or more of the original linguistic variables and therefore less comprehensible than the rules derived by earlier fuzzy inference systems.

At this point the question may again be raised, are these enhancements to fuzzy rule format justified? Unfortunately, as described in section 2.3.1, practically no meaningful information exists with which comparisons between the accuracy of simple Mamdani or Sugeno rules and the accuracy of advanced format rules such as those produced by SONFIN or the clustering algorithm of Kim, et al. (1997) can be made. For example, the results obtained by these particular two algorithms are compared to those of other methods (that derive simpler rules). Each algorithm is evaluated against other techniques on only one data set. These are respectively the very well known Box-Jenkins gas-furnace data and the equally well known Mackey-Glass time series. In addition, for the Box-Jenkins data only the training error is reported. All in all this makes it difficult to determine the generality of the reported improvements in accuracy.

In summary only a few observations can therefore be made. First, the natural language format of the Mamdani rule format was quickly updated to contain mathematical constructs such as the linear consequent function of Sugeno rules. Second, as far as the author is aware extremely few algorithms have been proposed that apart from conjunction, use further fuzzy language operators such as disjunction or negation to enhance the performance of derived fuzzy rules (see also section 3.1.3.3 for further information). Rather, those algorithms that have tried to build better fuzzy rules have increased the mathematical content of the fuzzy

rules, by for instance generating oriented hyperellipsoids (and in turn oriented membership functions).

In the context of the format of fuzzy rules it therefore seems that most, if not all, enhancements to fuzzy rules are in the form of mathematical constructs rather than fuzzy language constructs. In this regard the fuzzy modelling field is following the same path as was followed in crisp tree induction (described in section 3.2.2). In other words, instead of generating easily understood fuzzy concepts to reduce problem complexity, fuzzy rules are being adapted to be more accurate at the expense of human comprehensibility and intelligibility.

### 3.3 Discussion of Algorithmic Limitations

Before conclusions are drawn from the discussion presented in sections 3.1 and 3.2, a more in depth analysis will be made of the empirical results generated by Murthy and Salzberg (1995) as well Quinlan and Cameron-Jones (1995). Both these sets of results were generated to evaluate the efficacy of performing less greedy search than the simple greedy search methods typically used to induce decision trees or generate rules by the set covering approach.

In addition, results recently reported by Schmitz (1999) will also be examined. One of the studies performed by Schmitz investigated the effect of lookahead in the evolutionary construction of axis-parallel hyperellipsoids in single hidden layer neural networks. In these studies the single neural network output node consisted of a linear transfer function. As mentioned in section 2.3.6.2 such neural networks can be interpreted as a set of fuzzy rules. Therefore, even though the primary objective of Schmitz was not to build fuzzy rules, these particular results will provide additional insight into the usefulness of lookahead in rule construction.

The purpose of this study is twofold. The first purpose is to determine whether lookahead can in fact improve either the generalisation performance or reduce the complexity of the rules that are generated by the various algorithms. Model complexity is defined here as the number of rules used in the model. The second purpose is to determine how the quality of the data that was analysed by the various researchers influences the results obtained by the algorithms that were investigated.

### 3.3.1 Analysis of Data with a Low Exemplar to Attribute Ratio

Consider first the simulation performed by Freedman (1983). Although not directly applicable, the results obtained by Freedman provide some insight into the analysis of problems with many attributes but few data. Freedman first generated a data set consisting of one hundred exemplars each described by fifty-one attributes. All exemplars were independently drawn from the standard normal distribution. As stated by Freedman such data is pure noise. The fifty-first attribute was assumed to be the dependent variable, or output. Note that owing to the construction of the data, the output is independent of the remaining fifty attributes. This means that the  $R^2$  (a measure of the variance explained by the output) obtained from the regression should be insignificant by standard tests such as the F-test. Likewise, the regression coefficients should be insignificant, by the standard t-test.

Freedman analysed the data in two successive multivariate regression experiments. First, the regression was performed using all fifty attributes with results:  $R^2 = 0.50$ ,  $P = 0.53$ . Second, only those attributes whose coefficients reached a 25% significance level in the first experiment were used in a second experiment. The result of the second run was that the regression was highly significant ( $R^2 = 0.36$ ,  $P = 5 \times 10^{-4}$ ). Fourteen coefficients out of fifteen were significant at the 25% level. Six of the fifteen regression coefficients were significant at the 5% level. The results from the second experiment are very misleading since they appear to demonstrate a definite relationship between the fifty attributes and one output, i.e. between noise and noise. In summary, the results presented by Freedman show that standard statistical significance tests applied to regression models based on randomly generated data with many attributes can produce deceptive results.

### 3.3.2 Results Obtained by Advanced Rule Construction Methods

This section focuses on the experimental data used by Murthy and Salzberg (1995), and Quinlan and Cameron-Jones (1995) in their respective analyses. A summary of the relevant information is presented in table 3.2. Note that all the data sets that were considered by these authors are classification problems.

The first column in table 3.2 indicates which data set was investigated. The second column in table 3.2 gives three properties of each data set, viz. the number of exemplars, the number of attributes and the number of output classes. The density ratio column indicates the ratio of the number of exemplars over the number of attributes for a particular data set. The fourth and fifth columns present the relevant testing accuracy and model complexity results obtained by

Murthy and Salzberg. These results were obtained by performing five-fold cross-validation on the given set of data. The results after tree pruning for both greedy tree induction as well as one-level lookahead are given. Dashes indicate that the particular data set was not investigated. Similarly, the sixth and seventh columns summarise the relevant results obtained by Quinlan and Cameron-Jones for greedy and layered rule construction. These results are the average over 500 runs for each data set. For each run the data were randomly split into two equal subsets, one for training and one for testing purposes. The accuracies given in column six are testing accuracies.

Apart from the real-world data sets mentioned in table 3.2, Murthy and Salzberg (1995) also investigated a large number of synthetic data sets. Owing to the fact that these data had only two input attributes these results are considered too simple for further analysis.

Schmitz (1999) studied the effects of lookahead using three synthetic data sets and two real-world data sets. The relevant results of the experiments performed by Schmitz are summarised in table 3.3. All problems except the sonar problem are regression problems. The sonar problem is a classification problem. The “Features” column gives the number of data and input attributes, respectively. For all problems half the data were used for training and half for testing purposes, with the exception of the furnace data set. For this data set 75% of the data were used for training and the remainder for testing. The ratio of number of exemplars over input dimensionality is given in the third column of table 3.3. Columns four through seven give the testing accuracy of trained set of fuzzy rules. The level of lookahead is defined as the number of axis-parallel hyperellipsoids, in other words fuzzy rules, which are simultaneously constructed in addition to the current hyperellipsoids.



Data Set	Features	Density Ratio	Murthy and Salzberg (1995)		Quinlan and Cameron-Jones (1995)	
			Tree Accuracy (%) (Greedy   Lookahead)	Number of Leaves (Greedy   Lookahead)	Rule Accuracy (%) (Greedy   Layered)	Number of Rules (Greedy   Layered)
Promoters	106   57   2	1.9	-	-	72.6   75.4	8.3   5.5
Breast Cancer	286   9   2	3.2	72.8   69.4	4.71   13.5	71.2   71.2	43.0   29.4
Labor Negotiations	57   16   2	3.6	80.6   79.4	2.35   2.35	-	-
Hepatitis	155   19   2	8.2	78.8   77.5	3.53   4.71	81.9   80.9	14.3   10.4
Lymphography	148   18   4	8.2	71.3   71.4	8.24   8.83	77.9   81.1	14.4   10.4
Auto Insurance	205   24   6	8.5	-	-	68.6   68.9	33.4   18.7
Chess Endgame	551   39   2	14	-	-	89.3   89.7	44.0   28.9
Primary Tumor	339   17   21	20	-	-	41.7   41.5	59.8   53.3
Soybean	683   35   19	20	-	-	88.3   87.6	39.4   35.9
Glass Identification	214   10   7	21	65.6   65.0	14.7   13.5	63.8   65.9	27.3   18.1
Heart Disease	303   14   2	22	76.9   75.3	8.24   10.0	-	-
Voting (version 1)	435   16   2	27	95.0   95.0	5.30   3.83	94.3   94.4	14.3   10.9
Voting (version 2)	435   15   2	29	86.9   86.3	8.24   6.24	-	-
Credit Approval	690   15   2	46	-	-	83.3   83.6	58.5   31.7
Pima Diabetes	768   8   2	96	-	-	74.1   73.1	96.3   50.1

**Table 3.2 Predictive Classification Accuracy and Model Complexity Results**

Data Set	Features	Density Ratio	Level of Lookahead			
			None	2 Rules	4 Rules	6 Rules
Sonar	208   60	3.47	71.2	82.7	89.4	86.5
Mackey-Glass	100   4	25	99.89	99.91	99.08	99.93
Furnace	1692   10	169	65.1	70.6	72.8	71.9
kine8nh	2048   8	256	52.6	54.1	52.7	53.2
kine8nm	2048   8	256	76.4	78.7	81.4	82.3

**Table 3.3 Predictive Performance Using Different Levels of Lookahead**

Consider classification results reported by Murthy and Salzberg. It can be seen that the classification accuracy obtained by the greedy classification tree algorithm in all cases was either superior or comparable to that of the one-level lookahead algorithm. Murthy (1996, p.152) reports that these differences in accuracies are not statistically significant. In contrast, the layered search technique of Quinlan and Cameron-Jones obtains improved results in seven of the twelve problems investigated. The remaining results are either comparable or at worst 1% less than the classification accuracy obtained by the greedy algorithm. However, the author is of the opinion that the statistical technique used by Quinlan and Cameron-Jones to test the statistical significance of these results is unfortunately of questionable validity (see the footnote in section 3.1.3.2 for more information) and should be handled with care. A one-sided, paired t-test was therefore performed on only the twelve accuracy results given in column six of table 3.2. It showed that layered search generates marginally better results than simple greedy search with a 88.7% confidence level.

Next, consider the accuracy results reported by Schmitz (1999) and presented in table 3.3. A one-sided, paired t-test was again performed over all the data sets to determine whether lookahead gave improved predictive performance over no lookahead. Note that for each data set the best lookahead result was compared to the result for no lookahead. It was found that lookahead produced results that were significantly superior (94.7% confidence) to those obtained using no lookahead. Lookahead obtained better results for each of the problems considered.

In summary, results obtained from the literature indicate that a search methodology that is less greedy than simple greedy search is capable of generating accuracy results that are either comparable or significantly superior to those obtained by simple greedy search.

Consider now the complexity results (in terms of the number of tree leaves) reported by Murthy and Salzberg. These are presented in table 3.2 (column 5). Murthy and Salzberg (Murthy and Salzberg; 1995, p.153) state that lookahead (with tree pruning) does not produce

less complex decision trees than simple greedy search with pruning does. What is interesting to note in table 3.2 is that in three of the four cases where the density ratio is 8.2 or less, the tree model complexity obtained using lookahead is worse than that of simple greedy search. In contrast, in three of the four cases where the density ratio exceeds twenty, lookahead produces simpler trees.

The picture is clearer when one considers the complexity of the layered search method proposed by Quinlan and Cameron-Jones in comparison to the greedy rule construction technique. For each of the twelve problems investigated, the complexity of the rules derived using layered search is less than that of greedy search (the one-sided, paired t-test confidence level is 92.7%). In addition, the disparity between the model complexities is most pronounced for the two data sets (Credit Approval and Pima Diabetes) that have a density ratio exceeding 45. Furthermore, Quinlan and Cameron-Jones also compare the rule models derived by the two methods in terms of so-called theory size. The theory size for a particular rule model is the total number of antecedents used in the rule model. Using this measure, rule models derived by layered search are again significantly simpler than those obtained through greedy search (98.9% confidence).

In summary therefore results obtained from the literature indicate that less greedy search methods are capable of generating rule models that are significantly less complex than those derived by their greedy counterparts. This is especially true for problems where the density of data is relatively high.

### **3.3.3 Reasons for the Apparent Poor Performance of Advanced Rule Construction Techniques**

It is argued that the poor results (especially in terms of predictive accuracy) obtained by the one-level lookahead decision tree algorithm in particular, and the layered search method to a lesser extent, are not principally because of the incapability of such methods to generate better decision trees or rules. Rather, it is predominantly as a result of two factors or reasons.

The first is that the one-level lookahead method of Murthy and Salzberg is not tempered with additional search constraints. Such constraints are necessary to prevent the more flexible search technique from generating decision tree models that fit the training data well (better than greedy search methods do) but that do not generalise well to unseen data. The same argument has been put forward by Quinlan and Cameron-Jones (1995). They coin the phrase “oversearching” to describe the discovery by nongreedy search techniques of so-called

“fluke” models that fit the training data well but that generalise poorly. This argument is substantiated by the results presented by Quinlan and Cameron-Jones as well as those presented in section 3.3.2.

The second reason why advanced search techniques can struggle to obtain better results than simple greedy techniques is because of the poor choice of data sets that are typically examined in such comparative investigations. In particular, it is sometimes the case that the data of a particular problem are sparsely spread over the input space of the given problem. For example, Murthy and Salzberg examined a real-world “Labour Negotiations” data set consisting of 57 exemplars described by 16 attributes. It should be clear that these training exemplars are extremely sparsely distributed in the 16-dimensional attribute space of the problem. This makes it difficult for any modelling technique, including tree induction with lookahead, to discover the true underlying model represented by the data. This in turn means that it is difficult for nongreedy techniques to generate models that are appreciably better than those obtained with greedy methods. This argument is substantiated by the fact that the layered search method of Quinlan and Cameron-Jones obtained the greatest decrease in model complexity (number of rules and theory size), in comparison to simple greedy search, for the two data sets with the highest density ratio.

### 3.3.4 Conclusions

In the light of the above analysis as well as the discussion presented in sections 3.1 and 3.2, the following conclusions can be drawn with respect to the limitations of current rule construction techniques:

- Rule construction techniques that are not completely greedy and that are able to explicitly detect and utilise attribute interactions (typical in most chemical processes) during the search for a good set of rules can produce sets of rules that are significantly less complex than those obtained using greedy techniques. If both types of techniques generate rules using exactly the same representational language, reduced complexity implies improved rule comprehensibility and intelligibility. Furthermore, in most cases such rule sets have predictive accuracy that is comparable to the accuracy of rule sets obtained with greedy methods. A major problem with nongreedy search methods is the significant increase in computational complexity incurred by using such methods.

- The fragmentation and partitioning problems can have a significantly detrimental effect on both predictive performance as well as rule model comprehensibility. This is true in the decision tree field, where training data partitioning occurs, as well as for those set covering techniques that use a “separate and conquer” approach to find a good set of rules. It has been empirically shown that algorithms that do not suffer from data set fragmentation can obtain significantly more accurate and in some cases smaller rule models.
- In terms of both decision tree format as well as fuzzy rule format, the historical trend in the fields of tree induction and fuzzy modelling has been towards increasing the mathematical content of derived tree or rule models in order to improve predictive accuracy. In the author’s opinion, the increase in predictive accuracy (especially in terms of decision trees) usually does not justify the decrease in model comprehensibility and intelligibility.
- Rule set complexity can be further reduced if internal disjunction in the antecedent part is allowed and the search for such rules is guided using intelligent search restrictions (e.g. BEXA). Intelligent search restrictions can also significantly reduce the computational complexity involved in finding internally disjunctive rules (Theron and Cloete, 1996). As far as the author is aware, with one or two exceptions (e.g, Yuan and Zhuang, 1996) internal disjunction is not used in fuzzy modelling as a means of improving accuracy without resorting to the use of mathematical constructs in fuzzy rules. In addition, no fuzzy modelling techniques, including genetic algorithm approaches, use search restrictions such as those of BEXA to either guide the search towards a good set of fuzzy rules or to reduce the computational complexity of finding good rules.
- As shown in section 3.3.2 and by Quinlan and Cameron-Jones (1995), advanced rule constructions techniques that use less-than-greedy search are capable of generating rule models that are more accurate and significantly less complex than those obtained by their greedy counterparts. This is especially true for techniques (e.g. the layered search technique) that use search restrictions or heuristics to intelligently control the search for the best rule model.

### 3.3.5 A Final Note...

Readers familiar with modern fuzzy modelling techniques will question the need to spend so much time on the issue of nongreedy search methods, as has been done here. Modern fuzzy rule construction techniques such as the FuGeNeSys (Russo, 1998) genetic algorithm (described in section 2.3.5.3) or the GRBF network of Karayiannis and Mi (1997) (described in section 2.3.6.5) typically optimise the entire set of fuzzy rules simultaneously and use the entire set of data at every optimisation step. This is in contrast to the relatively greedy, stepwise methodology used by many crisp set covering methods (e.g. BEXA (Theron, 1994)) or the recursive partitioning methods used in decision tree construction. (Note that the latter method inherently constructs rules one at a time in a greedy fashion and in isolation from other rules.)

One of the aims of this fairly lengthy discussion of nongreedy search techniques is to make readers from the crisp decision tree and crisp set covering research fields aware that other research fields successfully utilise nongreedy search techniques to build “if...then...” rules. In the author’s opinion there is currently relatively little communication between the crisp rule construction community and other communities (such as fuzzy rule modelling), especially with regards to improved rule construction search methods.

In addition, the author found that relatively little work has been performed by researchers in the crisp rule building field to determine the merits and demerits of nongreedy search as a means of constructing rules. This discussion therefore has the secondary aim of comparing all of the more recent results that are available to the author with each other, and to in turn highlight possible areas of improvement.



# Chapter 4

## The Combinatorial Rule Assembler

In light of the conclusions presented at the end of chapter three, the Combinatorial Rule Assembler (CORA) algorithm is proposed. The CORA algorithm constructs models using the rules generated by an existing fuzzy rule construction technique in order to obtain a new set of rules with maximum predictive performance and improved human comprehensibility and intelligibility. The algorithm is predominantly designed to model regression problems but can also be used for classification problems.

The CORA algorithm is based upon two existing algorithms. The first is Fritzke's Growing Neural Gas (GNG) radial basis function neural network training algorithm (Fritzke, 1994a) and the second is the Reactive Tabu Search (RTS) algorithm (Battiti and Tecchiolli, 1994a). The CORA algorithm first constructs a set of fuzzy rules using the GNG algorithm. Thereafter the antecedent parts of these fuzzy rules are combined using the RTS combinatorial optimisation technique to form a smaller set of fuzzy rules. Finally, this second set of rules is simplified with the aim of reducing rule complexity (and thus improve human comprehensibility) without significantly affecting overall predictive performance.

The first part of this chapter describes the characteristic features of the neural network model that is generated by GNG training, such as automatic network size and structure identification, as well as topology preservation. Furthermore, the GNG algorithm can be used for both unsupervised attribute map learning and for supervised training of a radial basis function neural network. Those aspects of the supervised variant that are different from those of the unsupervised variant are also described, owing to their importance in the proposed combinatorial rule assembler algorithm.

As a precursor to the description of the RTS algorithm, two phenomena that are commonly found in the search space of nonlinear combinatorial optimisation problems are described. These phenomena are limit cycles and attractors. Subsequently the RTS methodology itself is described. Particular attention is paid to the so-called reaction mechanisms used by the RTS algorithm to handle the above-mentioned limit cycles and attractors.

The next part of the chapter presents reasons why the GNG and RTS algorithms were chosen over other fuzzy rule construction techniques (e.g. clustering techniques) or combinatorial optimisation methods (e.g. genetic algorithms).

The chapter ends with a discussion of the specific implementation details of the CORA algorithm. These include changes made to the GNG algorithm of Fritzke (1997), the way in which the combinatorial problem investigated by the RTS algorithm is defined and how the consequent of a fuzzy rule is calculated. The method by which membership functions are merged and simplified during fuzzy rule construction, as well as the criteria used to find the optimal size rule model, are also described. In addition, a description is given of the method by which the predicted output surface of the set of fuzzy rules is smoothed. Furthermore, this final chapter section describes how the number of trained and merged set of fuzzy rules is itself reduced, with the aim of deleting superfluous rules. Finally, the computational complexity of the CORA algorithm is discussed.

## 4.1 The Growing Neural Gas Algorithm

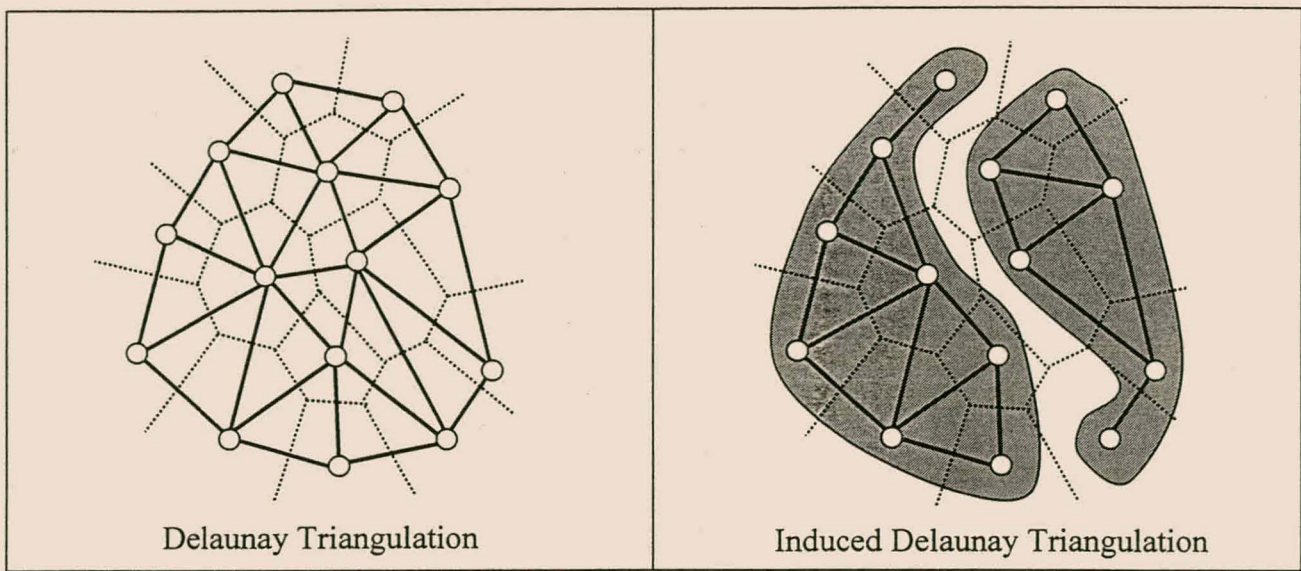
The Growing Neural Gas (GNG) algorithm (Fritzke, 1994a, 1995a) combines the growth mechanism of the Growing Cell Structures (GCS) algorithm (Fritzke, 1992, 1994b) with the topology generation of the Neural Gas (NG) algorithm (Martinetz and Schulten, 1991). The GNG algorithm (Fritzke, 1994a) has two variants for training neural networks. The first variant uses unsupervised, self-organising, topology preserving learning to automatically determine both the structure and size of the neural network model. This capability is a distinct advantage over other unsupervised, self-organising learning methods such as Kohonen's self-organising feature map (Kohonen, 1982). Kohonen's method requires a predetermined network size and structure.

Furthermore, both variants of the GNG algorithm construct a lateral connection structure between the nodes of the neural network model. The competitive Hebbian learning rule (Hebb, 1949; Martinetz, 1993) is used for this purpose. In contrast to precursors of the GNG algorithm, such as Fritzke's Growing Cell Structures algorithm (Fritzke, 1992, 1994b), the GNG algorithm can locally adapt the network architecture to the intrinsic dimensionality of the attribute

manifold. GNG network learning is thus able to generate a perfectly topology preserving attribute map, whatever the true input dimensionality of the training data.

Consider a graph (network)  $G$  with vertices (neural nodes)  $i$ ,  $1 \leq i \leq n$  and edges (lateral connections). Let  $M \subseteq \mathbb{R}^d$  be a given manifold of attributes in  $d$ -dimensional embedding space. Let  $S = \{c_1, \dots, c_n\}$  be a set of synaptic weight vectors, or pointers,  $c_i \in M$ , each of which is attached to a vertex  $i$  of the graph  $G$ . Let  $A \subset S \times S$  be the set of edges (lateral connections). Furthermore, let  $V_i = \{x \in \mathbb{R}^d \mid (\|x - c_i\| \leq \|x - c_j\|, 1 \leq j \leq n)\}$  be the Voronoi polyhedron belonging to  $c_i \in M$ .  $\|x - c_i\|$  is the activation of the  $i$ -th neural node with  $c_i$  as the centre of its receptive field upon presentation of an input signal  $x$ . Finally, let  $V_i^{(M)} = V_i \cap M$  be the masked Voronoi polyhedron of  $V_i$ ,  $1 \leq i \leq n$ . The definition of a perfectly topology preserving map is as follows (see also Martinetz and Schulten, 1994; Bruske and Sommer, 1995b).

- Two points  $c_i, c_j \in S$  are adjacent on  $M$  if their masked Voronoi polyhedra  $V_i^{(M)}, V_j^{(M)}$  are adjacent (have some boundary points in common), i.e.  $V_i^{(M)} \cap V_j^{(M)} \neq \emptyset$ .
- The induced Delaunay triangulation  $D_S^{(M)}$  of  $S$ , given the manifold  $M$ , is defined by the graph which connects two points  $c_i, c_j$  if and only if their masked Voronoi polyhedra  $V_i^{(M)}, V_j^{(M)}$  are adjacent.
- The set  $S$  is dense on  $M$  if for each  $v \in M$  the triangle  $\Delta(v, c_{i0}, c_{i1})$  formed by the pointer  $c_{i0}$ , which is closest to  $v$ , the point  $c_{i1}$ , which is second closest to  $v$ , and  $v$  itself lie completely on  $M$ .
- If the distribution of points  $c_i \in S$  is dense in  $M$ , then the graph  $G$  that is formed by the competitive Hebb rule is the induced Delaunay triangulation  $D_S^{(M)}$  of  $S$ , and hence forms a perfectly topology preserving map of  $M$ .



**Figure 4.1 Two Ways of Defining Proximity among a Set of Data Points**

Figure 4.1 illustrates the concepts of the Delaunay triangulation and the induced Delaunay triangulation in two dimensions. In the figure, the white bullets represent a subset of pointers of the above-mentioned set  $S$ . Each pointer is attached to a vertex of the graph  $G$ . The dotted lines represent the Voronoi diagram of the set of pointers. On the lefthand side of figure 4.1, the solid lines connecting the points represent the corresponding Delaunay triangulation (corresponding to a subset of  $A$ , the set of edges of  $G$ ). The Delaunay triangulation connects vertices that have neighbouring Voronoi polyhedra. The induced Delaunay triangulation, depicted on the righthand side of figure 4.1, is obtained by masking the original Delaunay triangulation with the data distribution (represented by the two grey areas) of the problem under consideration. In the induced Delaunay triangulation two pointers are connected by an edge if the common border of their Voronoi polyhedra lies at least partially in a region where there are data.

The supervised GNG variant is a combination of the above-mentioned, topology preserving<sup>1</sup>, self-organising learning method with a radial basis function neural network architecture for supervised learning. The radial basis function neural network has a single hidden layer and one

<sup>1</sup> Publications of the supervised GNG algorithm (Fritzke, 1994a, 1997) or its variants (Fritzke, 1995b) do not explicitly discuss the topology-preserving characteristics of the supervised GNG variant. Inspection of pictures presented in these publications that depict the topology of the trained radial basis function neural network's hidden layer indicate that there is topology preservation. It will therefore be assumed that the supervised GNG variant is topology preserving. In addition, adaptations (described in section 4.3.4.1) are made to the supervised GNG algorithm to bias the hidden layer topology towards that topology which would be obtained by the unsupervised GNG variant.



or more summation nodes in the output layer. The weights of the connections to the one or more output nodes are modifiable constants. The radial basis function neural network is trained by two learning methods in parallel. The supervised GNG algorithm is used to train the hidden layer of the radial basis function neural network and the delta rule (Widrow and Hoff, 1960; Stone, 1986) is used to train the weights to the one or more output nodes. The supervised GNG algorithm determines both the structure and size of the hidden layer.

### 4.1.1 The Unsupervised GNG Algorithm

The unsupervised GNG algorithm builds a perfectly topology preserving attribute map from a set of training exemplars according to a competitive Hebbian learning rule (Martinez and Schulten, 1991; Martinez, 1993). The competitive Hebbian learning method generates a subgraph of the Delaunay triangulation that is limited to those areas of the input space where training data are found. The goal of competitive learning here is the minimisation of the quantization error of the network model with respect to the set of training exemplars.

The network model consists of so-called cell nodes. Each cell node is characterised by a reference vector  $c \in \mathbb{R}^d$  where  $\mathbb{R}^d$  represents the  $d$ -dimensional attribute space. This reference vector indicates the position, or receptive field centre, of the cell node in input space. Furthermore, each cell node has an additional local error measure, or resource term, denoted  $r$ . The resource term is used to indicate which cell node has accumulated the most resource, or quantization error, during network training. This aspect is explained in more detail below.

Between each two cell nodes there can exist a so-called neighbourhood connection. Let  $A$  denote the set of internode connections. A neighbourhood connection is characterised by an age term, defined as  $a$ , but is otherwise unweighted and undirected. These internode connections have nothing to do with the connections that are found in typical neural networks (Rumelhart, et al., 1986). In particular, the connections do not indicate propagation of signals from one cell node to the next. Rather, the connections are used to indicate which cell nodes are in the topological neighbourhood of each other. This is true for both unsupervised and supervised learning.

**function initialise**  
 $S \leftarrow \{1, 2\}$   
Initialise  $c_1$  and  $c_2$  randomly from the set of training exemplars  
 $r_1 \leftarrow r_2 \leftarrow 0$   
 $A \leftarrow \emptyset$

*(create two cell nodes with centres  $c_1$  and  $c_2$ )*  
*(set resource of each cell node to zero)*  
*(no neighbourhood connections)*

Figure 4.2 Initialisation Function of the Unsupervised GNG Algorithm

The initialisation step of the unsupervised GNG algorithm, represented by figure 4.2<sup>2</sup>, creates two cell nodes. The reference vector of each cell node is initialised randomly from the set of training exemplars. The initialisation part is followed by the main loop that continues execution until one or more stopping criteria are met.

**procedure *unsupervised\_GNG***

***initialise***

*(see figure 4.2)*

**repeat**

Randomly select  $x$  from the set of training exemplars

$w(x) \leftarrow \arg \min \|x - c_i\|$  for all  $c_i \in S$  *(determine winner and runner up)*

$s(x) \leftarrow \arg \min \|x - c_i\|$  for all  $c_i \in (S - w(x))$

**if**  $(w(x), s(x)) \notin A$  **then** *(if connection does not exist, create one)*

$A \leftarrow A \cup \{(w(x), s(x))\}$

$a_{(w(x), s(x))} \leftarrow 0$

$r_w \leftarrow r_w + \|x - c_w\|^2$  *( $r_w$  = resource term of  $w(x)$ )*

$c_w \leftarrow c_w + \varepsilon_w(x - c_w)$  *( $c_w$  = reference vector of  $w(x)$ )*

**for all**  $i \in N_w$  *( $N_w$  = direct topological neighbours of  $w(x)$ )*

$c_i \leftarrow c_i + \varepsilon_n(x - c_i)$

**for all**  $(w(x), i) \in A$

$a_{(w(x), i)} \leftarrow a_{(w(x), i)} + 1$

**if**  $a_{(w(x), i)} > a_{max}$  **then** *(delete connections that are older than  $a_{max}$ )*

Delete  $(w(x), i)$  from  $A$

**for all**  $i \in S$

If cell node  $i$  has no emanating connections then delete it

**if**  $(n_x \bmod \lambda) = 0$  **then** *( $n_x$  is the number of exemplars seen thus far)*

**insert\_cell\_node** *(see figure 4.4)*

**for all**  $i \in S$

$r_i \leftarrow r_i(1 - \beta)$

**until** quantization error exceeds threshold or maximum iteration is reached

**Figure 4.3 Main Unsupervised GNG Procedure**

Figure 4.3 describes the main part of the unsupervised GNG algorithm. Given a training exemplar  $x$ , the winner  $w(x)$  among the existing cell nodes of the network is defined as the node with the nearest reference vector. The Euclidean norm, represented by  $\|\cdot\|$ , is used to determine the distance between the reference vector  $c$  of each cell node and the training exemplar  $x$ . The runner up cell node that is second closest to  $x$  is defined as  $s(x)$ .

<sup>2</sup> In each of these figures conditional (e.g. if...then...) and loop (e.g. repeat...until...) operators are displayed in boldface. The  $\leftarrow$  symbol represents the assignment operator. The statement  $x \leftarrow y$  therefore means that variable  $x$  is assigned the value of variable  $y$ . Functions that represent compound statements are displayed in italicised boldface. Regular text summarises compound statements. Comments are italicised and enclosed in parentheses.



At the start of the iterative process, the algorithm first determines which two cell nodes are closest to a randomly chosen training exemplar. If nonexistent, a neighbourhood connection is created between them. Thereafter the squared Euclidean distance in attribute space between the current training exemplar  $x$  and the winner  $w(x)$  is added to the resource term  $r_w$  of  $w(x)$ . The next step of the algorithm adapts the reference vectors of  $w(x)$  and the set  $N_w$  of its direct topological neighbours by user-defined fractions  $\varepsilon_w$  and  $\varepsilon_n$ , respectively. The age of each of the lateral connections of  $w(x)$  is then incremented. Those connections with ages greater than the user-defined threshold  $a_{max}$  are deleted. If this results in cell nodes having no emanating connections, these nodes are also deleted. If the number of training exemplars, defined as  $n_x$ , seen by the network is an integer multiple of a user-defined parameter  $\lambda$ , a new cell node is inserted. The final step of the loop decreases the resource term  $r_i$  ( $1 \leq i \leq n$ ) of each cell node by a user-defined fraction  $\beta$ . This operation is performed to stress the importance of recently occurred errors and to ensure that the magnitude of  $r_i$  does not exceed the bounds of the numeric system of the computer on which the algorithm is programmed and executed.

**function insert\_cell\_node**

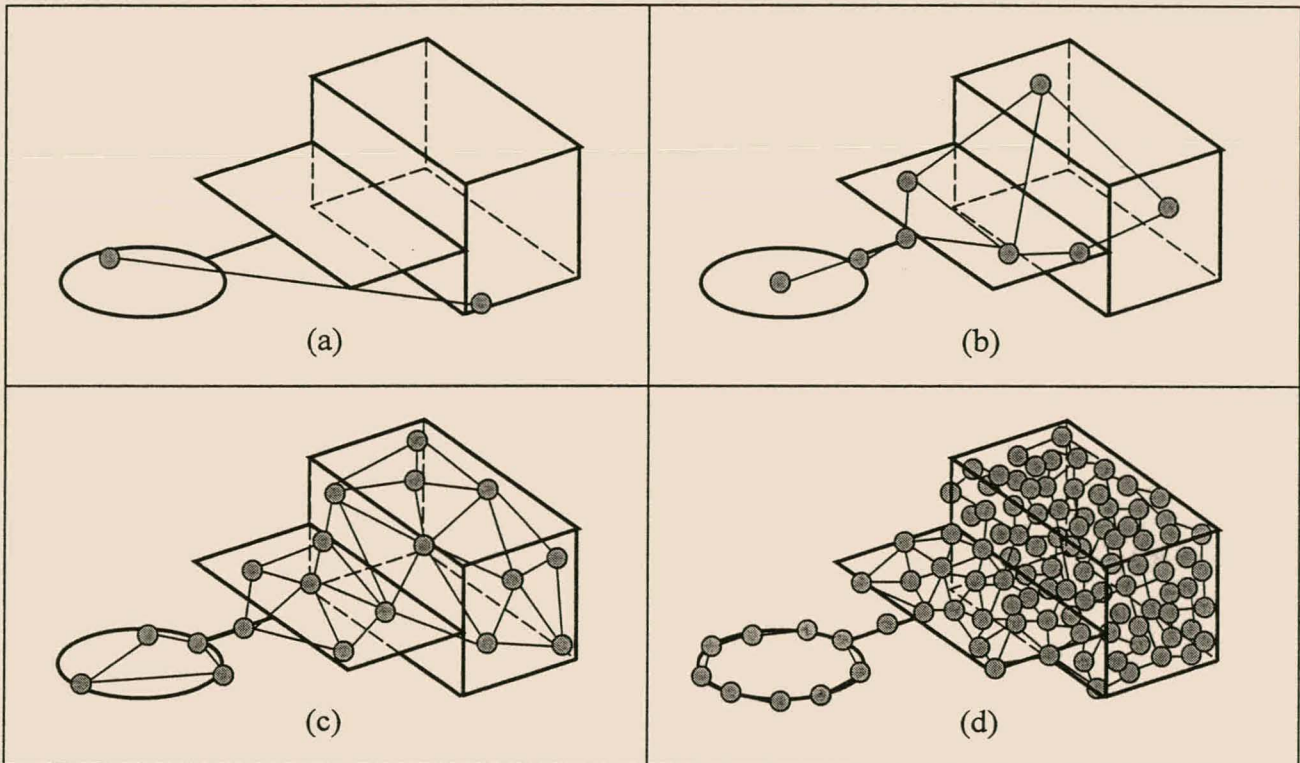
$q \leftarrow \arg \max r_i \text{ for all } i \in S$	<i>(determine max. resource node)</i>
$t \leftarrow \arg \max r_i \text{ for all } i \in N_q$	<i>(<math>N_q = \text{set of direct topological neighbours of } q</math>)</i>
$S \leftarrow S \cup \{u\}$	<i>(create cell node <math>u</math> with centre <math>c_u</math>)</i>
$c_u \leftarrow 0.5(c_q + c_t)$	<i>(interpolate reference vector)</i>
$A \leftarrow A \cup \{(q, u)\}$	<i>(insert neighbourhood connections)</i>
$A \leftarrow A \cup \{(t, u)\}$	
$a_{(q, u)} \leftarrow 0$	<i>(set connection ages)</i>
$a_{(t, u)} \leftarrow 0$	
Delete $(q, t)$ from $A$	<i>(remove neighbourhood connection)</i>
$r_q \leftarrow r_q(1 - \alpha)$	
$r_t \leftarrow r_t(1 - \alpha)$	
$r_u \leftarrow 0.5(r_q + r_t)$	<i>(interpolate resource)</i>

**Figure 4.4 Unsupervised GNG Cell Node Insertion Function**

Figure 4.4 describes the function used to insert a new cell node into the network. The function first determines which cell node  $q$  and direct topological neighbour of  $q$ , cell node  $t$ , has the maximum accumulated resource. A new cell node  $u$  is then created. The centre  $c_u$  of  $u$  is interpolated from those of  $q$  and  $t$ .  $u$  is then inserted between  $q$  and  $t$ , and the neighbourhood connection structure is updated to reflect this. Thereafter the resource term of each of  $q$  and  $t$  is

decreased by a user-defined fraction  $\alpha$ . Finally, the resource  $r_u$  of  $u$  is interpolated from those of  $q$  and  $t$ .

Figure 4.5(a) to figure 4.5(d) illustrates how the unsupervised GNG algorithm sequentially updates the neural network by both adapting the network structure and increasing the network size in order to best approximate an attribute manifold. The manifold depicted in each of the four quadrants of figure 4.5 consists of a two-dimensional circle attached to a two-dimensional plate by a one-dimensional line. The plate in turn is attached to a three-dimensional box. A cell node is represented by a grey bullet. Lines between cell nodes are internode connections. As can be seen from the figure, as new cell nodes are added to the initial two (figure 4.5(a)), the network structure adapts to the manifold's intrinsic dimensionality, forming a topology-preserving map.



**Figure 4.5 Unsupervised Learning of an Attribute Manifold**

Finally, inspection of the algorithm shows that the ageing of the neighbourhood connection structure is used to delete cell nodes that are not winning any training data. These deleted cell nodes can then be reallocated elsewhere in the network to better approximate the attribute manifold. In the algorithm considered here, a user-defined maximum number of cell nodes can be utilised by the GNG algorithm. In other words, once the maximum number of cell nodes has been allocated to the network, no more nodes are inserted unless one or more cell nodes are deleted during network adaptation.

### 4.1.2 The Supervised GNG Algorithm

This section describes how the supervised GNG algorithm is used to learn the topological structure and size of the hidden layer of a radial basis function neural network. The hidden layer of the radial basis function neural network consists of cell nodes, each characterised by a reference vector  $c \in \mathbb{R}^d$  as well as a local error measure, or resource term  $r$ . Instead of accumulating quantization error as in unsupervised learning, the resource term stores the accumulated classification error or regression error of the cell node. The error in both cases is calculated as the summed, squared difference between the network output and the true output of the exemplar under consideration. The aim of storing regression or classification error is to identify cell nodes in regions of attribute space where the radial basis function neural network performs poorly. This information is used to update existing cell nodes and to decide where new cell nodes are to be inserted in the attribute space.

Each cell node also has an associated kernel function. The kernel function chosen here is the hyperspherical Gaussian (described in section 2.1.2.2). The centre of the Gaussian is taken to be identical to the reference vector  $c$  of the cell node. The width, or variance, of the Gaussian is defined as  $\sigma$ . Since the Gaussian is hyperspherical, its width is the same in each attribute dimension.

As before, there can exist a neighbourhood connection with age  $a$  between each pair of cell nodes in the hidden layer of the radial basis function neural network. These internode connections are not used for signal propagation through the neural network. Instead, the connections from the hidden layer to the output layer are used for signal propagation. These interlayer connections have modifiable constants as weights. There is a single summation node in the output layer, although the number of nodes can be increased for multidimensional output problems. The network output is therefore the weighted sum of all the input signals to the output node.

The manner in which the winner cell node  $w(x)$  and the runner up cell node  $s(x)$  for a particular training exemplar  $x$  are chosen is the same as that used for unsupervised training. The Euclidean norm is again used to determine the distance between the reference vector  $c$  of each cell node and the training exemplar  $x$ . The complete supervised GNG algorithm (Fritzke, 1997) is described in figures 4.6 through 4.8.

**function initialise**

$S \leftarrow \{1, 2\}$	<i>(create two cell nodes with centres <math>c_1</math> and <math>c_2</math>)</i>
Initialise $c_1$ and $c_2$ randomly from the set of training exemplars	
$r_1 \leftarrow r_2 \leftarrow 0$	<i>(set resource of each cell node to zero)</i>
$A \leftarrow \{(1, 2)\}$	<i>(connect cell node 1 and 2).</i>
$a \leftarrow 0$	
$l_{total} \leftarrow 0$	<i>(set total length of connections to zero)</i>
$n \leftarrow 0$	
<b>for all</b> $(1, i) \in A$	<i>(determine total length of connections)</i>
$l_{total} = l_{total} + \ c_1 - c_i\ $	
$n \leftarrow n + 1$	<i>(count number of connections)</i>
$\sigma_1 \leftarrow \tau \times l_{total} / n$	<i>(Gaussian width is <math>\tau</math> times the mean length)</i>
$\sigma_2 \leftarrow \sigma_1$	<i>(Gaussian widths of 1 and 2 are identical)</i>

**Figure 4.6 Initialisation Function of Supervised GNG**

The initialisation function of the supervised GNG algorithm is described in figure 4.6. As before, two cell nodes are first created and initialised. Unlike the unsupervised version of the GNG algorithm, a connection is created between the two cell nodes to allow the width of the Gaussian of each cell node to be determined<sup>3</sup>.

The main loop of the supervised GNG algorithm (figure 4.7) is identical to that of the unsupervised version (figure 4.3) in all but three respects. The first is that instead of the quantization error, the summed, squared difference between the true output of the current training exemplar  $x$  and the output of the radial basis function neural network, with respect to  $x$ , is added to the resource  $r_w$  of the winner cell node  $w(x)$ . The second difference is that the weights of the connections from the hidden layer to the output layer of the radial basis function neural network are updated using the delta rule (Stone, 1986). This step is performed for each training exemplar  $x$  seen by the network. Note that in the initialisation step of the radial basis function neural network these weights are assigned random values in the range  $[0,1]$ . The third difference between the main loops of the two GNG variants is that different stopping criteria are used to exit the main loop. One criterion that has been used successfully in supervised training is the maximisation of performance on a validation set of data (Fritzke, 1997).

<sup>3</sup> Fritzke (1997) states that the width of the Gaussian of a cell node is calculated as the fraction  $\tau$  of the mean length of all connections emanating from the cell node. The first two cell nodes initially have no connections. The author has therefore adapted the supervised GNG algorithm described in Fritzke (1997) to first create a connection between the first two cell nodes and thereafter initialise the widths of the respective Gaussians as mentioned above.



**procedure supervised\_GNG****initialise**

(see figure 4.6)

**repeat**Randomly select  $x$  from the set of training exemplars $w(x) \leftarrow \arg \min \|x - c_i\|$  for all  $c_i \in S$  $s(x) \leftarrow \arg \min \|x - c_i\|$  for all  $c_i \in (S - w(x))$ **if**  $(w(x), s(x)) \notin A$  **then** $A \leftarrow A \cup \{(w(x), s(x))\}$  $a_{(w(x), s(x))} \leftarrow 0$  $r_w \leftarrow r_w + \|y - y'\|^2$  $(y = \text{true output of training exemplar } x)$  $c_w \leftarrow c_w + \varepsilon_w(x - c_w)$  $(y' = \text{network output for training exemplar } x)$ **for all**  $i \in N_w$  $(N_w = \text{direct topological neighbours of } w(x))$  $c_i \leftarrow c_i + \varepsilon_n(x - c_i)$ **for all**  $(w(x), i) \in A$  $a_{(w(x), i)} \leftarrow a_{(w(x), i)} + 1$ **if**  $a_{(w(x), i)} > a_{max}$  **then**Delete  $(w(x), i)$  from  $A$ **for all**  $i \in S$ If cell node  $i$  has no emanating connections then delete it**if**  $(n_x \bmod \lambda) = 0$  **then** $(n_x \text{ is the number of exemplars seen thus far})$ **insert\_cell\_node**

(see figure 4.8)

**for all**  $i \in S$  $r_i \leftarrow r_i(1 - \beta)$ 

Update all connection weights to the radial basis function neural network output layer

**until** maximum performance on a validation set is obtained or maximum iterations completed**Figure 4.7 Main Procedure of the Supervised GNG Algorithm**

The *insert\_cell\_node* function of the supervised GNG algorithm, described in figure 4.8, is identical to that of the unsupervised version (figure 4.4) in all but one respect. The supervised version includes a further adaptation step after the new cell node  $u$  has been inserted into the radial basis function neural network hidden layer and the connection structure has been updated. This step determines the width of the Gaussian of  $u$  and updates the widths of both  $q$  and  $t$  to accommodate  $u$ <sup>4</sup>. In particular, each cell node  $q$ ,  $t$  and  $u$  is considered in turn. The width of the Gaussian of the given cell node is calculated as the fraction  $\tau$  of the mean length of all connections emanating from the particular cell node.

<sup>4</sup> Fritzke (1997) does not explicitly state which cell nodes have the width parameter of their respective Gaussians updated when a new cell node is inserted into the radial basis function neural network's hidden layer. Furthermore, it is not explained whether Gaussian width updating occurs only when a new cell node is inserted or, for example, whenever a new training exemplar is presented to the radial basis function neural network. The algorithm described here is the one used in this dissertation.

**function insert\_cell\_node**

```

 $q \leftarrow \arg \max r_i \text{ for all } i \in S$ 
 $t \leftarrow \arg \max r_i \text{ for all } i \in N_q$  ( $N_q = \text{direct topological neighbours of } q$ )
 $S \leftarrow S \cup \{u\}$ 
 $c_u \leftarrow 0.5(c_q + c_t)$ 
 $A \leftarrow A \cup \{(q, u)\}$ 
 $A \leftarrow A \cup \{(t, u)\}$ 
 $a_{(q, u)} \leftarrow 0$ 
 $a_{(t, u)} \leftarrow 0$ 
Delete  $(q, t)$  from  $A$ 
for all  $i \in \{q, t, u\}$  (consider each cell node  $q, t$  and  $u$  in turn)
     $l_{total} \leftarrow 0$ 
     $n \leftarrow 0$ 
    for all  $(i, j) \in A$  (determine total length of connections of  $i$ )
         $l_{total} = l_{total} + \|c_i - c_j\|$ 
         $n \leftarrow n + 1$ 
     $\sigma_i \leftarrow \tau \times l_{total} / n$  (update Gaussian width of cell node  $i$ )
 $r_q \leftarrow r_q(1 - \alpha)$ 
 $r_t \leftarrow r_t(1 - \alpha)$ 
 $r_u \leftarrow 0.5(r_q + r_t)$ 

```

**Figure 4.8 Cell Node Insertion Function of the GNG Supervised Algorithm**

## 4.2 The Reactive Tabu Search

### 4.2.1 Limit Cycle and Attractor Detection and Avoidance

The Reactive Tabu Search (RTS) combinatorial optimisation technique (Battiti and Tecchiolli, 1994a; Battiti, 1996) is based on the principles of tabu search (Glover, 1989, 1990; Reeves, 1993). Tabu search is a metaheuristic method that combines a modified hill-climbing procedure based on a set of elementary moves with a set of heuristics. The purpose of the heuristics is to avoid premature convergence at suboptimal solutions and to prevent the occurrence of cycles in the search pattern. This goal is realised by utilising a list of so-called tabu, or prohibited, moves that are derived from the recent history of the search process. These moves are typically the inverses of moves that have been executed in the most recent part of the search.

The underlying assumption of this tabu list scheme is that suboptimal solutions found by the hill-climbing component are better starting solutions than randomly chosen ones. This assumption has two conditions. These conditions are firstly that local optima must not become attractors of the search dynamics induced by the algorithm and secondly that limit cycles do not arise. Analogous to chaotic systems (Glendinning, 1994), a limit cycle for a nonlinear search process is



a search trajectory that follows a closed curve with a fixed period within the search space. In other words, the trajectory endlessly repeats a sequence of suboptimal solutions.

Local optima are one possible form of attractor of search techniques that use hill climbing, viz. gradient descent techniques, to look for the global optimum. Such techniques will move to the same fixed point, or suboptimal solution, from a number of different starting positions if the search trajectory becomes trapped in the attraction basin of a local optimum. A second form of attractor is observed where the search trajectory is confined to a limited portion of the total search space. In this instance, unlike a limit cycle, the search trajectory does not follow a curve with a fixed period but still remains within a confined portion of the total search space.

The RTS scheme adapts the size of the list of tabu moves to avoid limit cycles. In particular, the RTS scheme has a fast reaction mechanism that increases the tabu list size if limit cycles are detected. Long lists of prohibited moves cause avoidance of cycles but constrain the search for optimal solutions. The fast reaction mechanism is therefore accompanied by a slower size reduction mechanism that reduces the list size when the search trajectory is in a region of the search space that does not exhibit limit cycles.

A further long-term diversification mechanism is enforced by the RTS scheme when an attractor in the search space is encountered. If an attractor is detected the RTS scheme executes an escape procedure to force the search trajectory out of the discovered attraction basin. In the following section both the list-size adaptation mechanism as well as the mechanism to escape attractors in the search space are discussed in more detail.

### 4.2.2 Summary of RTS Scheme

Consider a combinatorial optimisation problem with a set  $F$  of feasible solutions with finite cardinality and a cost function  $E$ . Without loss of generality assume that  $E$  needs to be maximised (for a minimisation problem take the negative of  $E$ ). The neighbourhood  $N(f)$  of a particular solution  $f$  is defined as the set of solutions that can be obtained by applying any one of a set of elementary moves  $M$  to  $f$ , i.e.  $N(f) = \{g \in F \mid (g = \mu(f), \mu \in M)\}$ . Consider the case where  $F$  is the set of all binary strings with finite length  $L$ . The elementary moves  $\mu_i$  ( $1 \leq i \leq L$ ) change the  $i$ -th bit of the string  $f = [f_1, \dots, f_i, \dots, f_L]$  to  $[f_1, \dots, \bar{f}_i, \dots, f_L]$  where  $\bar{f}_i$  is the negation of the  $i$ -th bit, i.e.  $f_i \equiv (1 - f_i)$ .

The combinatorial search starts from a randomly generated initial configuration  $f^{(0)}$  (superscripts with parentheses signify quantities that depend on the search iteration). An iterative process is then followed. At a given iteration  $t$  of the search, the set of moves  $M$  is partitioned into the set

$T^{(t)}$  of tabu, i.e. prohibited, moves and the set  $A^{(t)}$  of admissible moves. The set of prohibited moves is defined as  $T^{(t)} = \{\mu \in M \text{ such that its most recent use has been at time } \Lambda(\mu) \geq (t - T^{(t)})\}$ .  $\Lambda(\mu)$  is defined in section 4.2.3. Consequently the set of admissible moves is  $A^{(t)} = \{\mu \in M \text{ such that its most recent use has been at time } \Lambda(\mu) < (t - T^{(t)})\}$ .

At  $t = 0$  the partitioning is  $A^{(0)} = M$  and  $T^{(0)} = \emptyset$ , viz. all moves are admissible. The modified hill climbing component then selects the best move  $\mu^{(t)}$  from  $A^{(t)}$ . The best move is the move that maximises  $E$ . Note that the best move is executed even if  $E$  decreases with respect to its previous value. The aim of this modification is to allow the hill climbing procedure to exit from local maxima. At the given iteration  $t$ , the successor to the current solution is obtained by applying the best move to the current solution:  $f^{(t+1)} = \mu^{(t)}(f^{(t)})$  where  $\mu^{(t)} = \arg \max E(v(f^{(t)}))$  for all  $v \in A^{(t)}$ . If more than one move results in the same maximum value of  $E$ , the move that is applied is randomly selected from those with the same  $E$  value. The set of solutions encountered by the above discrete search process is called a trajectory.

After the execution of the move  $\mu^{(t)}$ , both  $A^{(t)}$  and  $T^{(t)}$  are updated to prevent limit cycles from occurring and to ensure that  $A^{(t)}$  is not empty, i.e.  $A^{(t)} \neq \emptyset$ . The latter condition is enforced to allow the search process to continue indefinitely (if so desired). A simple example of a limit cycle that may occur if the move  $\mu^{(t)}$  is not prohibited in the next iteration is when the value of  $E$  of the new solution is less than that of the previously found solution, i.e.  $E(\mu^{(t)}(f^{(t)})) < E(f^{(t)})$ . In this instance, the modified hill-climbing component in the next iteration would choose the reverse move of  $\mu^{(t)}$ . This would result in the original solution being obtained again.

As mentioned in the previous section the size, defined as  $S$ , of the set  $T^{(t)}$  of prohibited moves is dynamically adjusted, according to the local structure of the optimisation problem, to prevent such limit cycles. In detail,  $S$  must be greater than  $(R/2) - 1$  to make cycles of length  $R$  unattainable. The prohibition on moves must be cancelled again after a certain number of iterations because the moves can be necessary to reach the global maximum in a later stage of the search. In the RTS scheme the most recent iteration when each move  $\mu_i$  has been applied and each solution  $f^{(t)}$  encountered in the search trajectory is stored in memory with the most recent time that it was visited.

### 4.2.3 The RTS Algorithm

The structure of the RTS algorithm for the binary string combinatorial optimisation problem described above is summarised in figures 4.9 through 4.13. The following variables are defined.

$S^{(t)}$  is the size of the set of prohibited moves at iteration  $t$ .  $t_S$  stores the last time  $S$  was modified.  $C$  represents the set of often-repeated solutions.  $R_{avg}$  is a moving average of the repetition interval.  $f_b$  is the best solution found thus far.  $E_b$  is the value of the cost function for  $f_b$ . Furthermore, define the functions  $\Lambda(\mu)$  as the last iteration when  $\mu$  has been used,  $\Pi(f)$  as the last iteration when  $f$  was encountered and  $\Phi(f)$  as the number of times  $f$  has been visited in the search trajectory.  $\Lambda(\mu) = -\infty$  if  $\mu$  has never been used.  $\Pi(f) = -\infty$  if  $f$  has never been encountered.  $\Phi(f) = 0$  for all solutions at the start of the search process.

**function *initialise***

$t \leftarrow 0$	(iteration counter)
$t_S \leftarrow 0$	(last time $S$ was changed)
$S^{(0)} \leftarrow 1$	(tabu list size)
$C \leftarrow \emptyset$	(set of often repeated solutions)
$R_{avg} \leftarrow 1$	(moving average of repetition interval)
$f^{(0)} \leftarrow \text{random } f \in F$	(randomly generate initial solution)
$f_b \leftarrow f^{(0)}$	(store best solution)
$E_b \leftarrow E(f^{(0)})$	(store best cost function value)

**Figure 4.9 RTS Initialisation Function**

Figure 4.9<sup>5</sup> summaries the initialisation function of the RTS algorithm. The tabu list size  $S$  is initialised with a small value (e.g. 1) and then adapted in reaction to the occurrence of repetitions in the search trajectory. The starting solution is generated randomly and stored as the current best solution.

<sup>5</sup> The pseudocode of the RTS algorithm is presented in the same notation used for the pseudocode description of the GNG algorithm. In addition, numerical constants (e.g. CHAOS) and Boolean values (e.g. DO\_NOT\_ESCAPE) are fully capitalised.

**procedure RTS****initialise**

(see figure 4.9)

**repeat**escape  $\leftarrow$  *check\_for\_repetitions*( $f^{(t)}$ ) (see figure 4.11)**if** escape = DO\_NOT\_ESCAPE **then** $\mu \leftarrow$  *best\_move*

(see figure 4.12)

 $f^{(t+1)} \leftarrow \mu(f^{(t)})$  $\Lambda(\mu) \leftarrow t$ ( $A^{(t)}$  and  $T^{(t)}$  are implicitly updated) $t \leftarrow t + 1$ **if**  $E(f^{(t)}) > E_b$  **then** $E_b \leftarrow E(f^{(t)})$  $f_b \leftarrow f^{(t)}$ **else***diversify\_search*

(see figure 4.11)

**until**  $E_b$  satisfies the stopping criterion or the maximum iteration is reached**Figure 4.10 Main RTS Procedure**

The main loop of the RTS algorithm is described in figure 4.10. The loop iterates until  $E_b$  reaches an acceptable level or the user-defined maximum number of iterations has been completed. The loop starts by calling the function *check\_for\_repetitions* (figure 4.11). This function compares the current solution to those stored in memory that have been visited previously. Note that it is possible for a previously visited solution not to be stored in memory. This can happen if there is insufficient memory to store all solutions or the RTS algorithm has cleared the memory content (in particular see the function *diversify\_search* (figure 4.13)).

If *check\_for\_repetitions* returns the Boolean ESCAPE a diversification procedure is executed. This procedure is described by the *diversify\_search* function (figure 4.13). Alternatively, if the Boolean DO\_NOT\_ESCAPE is returned then the next move in the search trajectory is selected by the function *best\_move* (figure 4.12) and applied to the current solution  $f^{(t)}$ . If the cost function value of the new solution exceeds  $E_b$  then  $f_b$  and  $E_b$  are updated to store the new best solution information. The best solution found thus far is stored explicitly because the RTS algorithm clears the memory storing the set of previously visited solutions when a diversification procedure (see function *diversify\_search*) is executed.

## 4.2.4 The RTS Reactive Mechanisms

```

function check_for_repetitions(f)

  if  $\Pi(f) > -\infty$  then
     $R \leftarrow t - \Pi(f)$                                 (store time interval between visits to f)
     $\Pi(f) \leftarrow t$ 
     $\Phi(f) \leftarrow \Phi(f) + 1$ 
    if  $\Phi(f) > \text{REP}$  then
       $C \leftarrow C \cup f$                                 (add f to the set of often-repeated solutions)
      if  $|C| > \text{CHAOS}$  then
         $C \leftarrow \emptyset$ 
        return ESCAPE
      if  $R < 2(L - 1)$  then
         $R_{\text{avg}} \leftarrow 0.1 \times R + 0.9 \times R_{\text{avg}}$ 
         $S^{(t+1)} = \min(S^{(t)} \times \text{INCREASE}, L - 2)$ 
         $t_S \leftarrow t$ 
    else                                                (if the solution is not found, store it)
       $\Pi(f) \leftarrow t$ 
       $\Phi(f) \leftarrow 1$ 
    if  $(t - t_S) > R_{\text{avg}}$  then
       $S^{(t+1)} \leftarrow \max(S^{(t)} \times \text{DECREASE}, 1)$ 
       $t_S \leftarrow t$ 

  return DO_NOT_ESCAPE

```

**Figure 4.11** The RTS *check\_for\_repetitions* Function

The reaction mechanisms of the RTS algorithm modify the search trajectory to minimise the negative effects of limit cycles and attractors. The modifications are based on the past history of the search and cause possible changes to  $S^{(t)}$  or the execution of a diversifying procedure. The former modification is performed by the function *check\_for\_repetitions* (figure 4.11) and the latter by the *diversify\_search* function (figure 4.13).

*check\_for\_repetitions* compares the current solution  $f$  with those solutions that have been visited previously and are stored in memory. If  $f$  is in memory the last time that it was visited,  $\Pi(f)$ , and the repetition counter  $\Phi(f)$  are updated. If the repetition count of  $f$  is greater than the user-defined threshold REP,  $f$  is added to the set  $C$  of often-repeated solutions. If the magnitude of  $C$ ,  $|C|$ , is greater than the user-defined threshold CHAOS, the function immediately returns the Boolean value ESCAPE.

If the repetition interval  $R$  is sufficiently short ( $R < 2(L - 1)$ ) for a binary string of length  $L$  limit cycles are discouraged by increasing  $S^{(t)}$  by a constant factor INCREASE. After a number  $\phi$  of

such fast reactions, the geometric increase (proportional to INCREASE<sup>9</sup>) will be sufficient to break any limit cycle. At least two moves must always be admissible to prevent the RTS algorithm from being unable to continue (Battiti and Tecchiolli, 1995b) and therefore an upper bound of  $L - 2$  is set on  $S^{(t)}$ .

A slower, long term reaction mechanism decreases  $S^{(t)}$ . Battiti and Tecchiolli (1995b) state that if this is not done then  $S^{(t)}$  will remain large after a phase of the search containing many repetitions, even in later phases, when a smaller value of  $S^{(t)}$  would be sufficient to avoid short limit cycles.  $S^{(t)}$  is therefore multiplied by the factor DECREASE if it remains constant for a number of iterations greater than  $R_{avg}$ , the moving average of repetition intervals.

**function *best\_move***

```

if  $|A^{(t)}| < 2$  then
     $S^{(t)} \leftarrow L - 2$ 
     $t_S \leftarrow t$ 
 $\mu \leftarrow \arg \max E(v(f^{(t)}))$  for all  $v \in A^{(t)}$            (find best admissible move)
if ASPIRATION then
     $\gamma \leftarrow \arg \max E(v(f^{(t)}))$  for all  $v \in T^{(t)}$    (find best tabu move)
if  $E(v(f^{(t)})) > E_b$  and  $E(v(f^{(t)})) > E(\mu(f^{(t)}))$  then
     $\mu \leftarrow \gamma$                                            (update if best ever solution)
return  $\mu$ 

```

**Figure 4.12 The *best\_move* Function of the RTS Technique**

The first part of the function *best\_move*, described in figure 4.12, checks that  $A^{(t)}$  contains at least two admissible moves to ensure that the search process can continue. Thereafter each admissible move is tested on the current solution. The newly generated solution with the greatest  $E$  value is stored in  $\mu$ . If the Boolean variable ASPIRATION is set to true then those moves that are currently tabu are also evaluated. If any of these moves generate a solution that has a greater  $E$  value than both that of the solution produced by  $\mu$  and  $E_b$ , then  $\mu$  is replaced by this move. The relaxation of the tabu status of this move is permissible since it is clear that repetitions are avoided (the new solution has never been seen before).

If the value of escape in the RTS algorithm main loop (figure 4.10) is ESCAPE then the *diversify\_search* function is called. This function is described in figure 4.13. The fast and slow reaction mechanisms of *check\_for\_repetitions* (figure 4.10) that adapt  $S^{(t)}$  are in some cases not sufficient to guarantee that the search trajectory is not confined within a limited portion of the search space. When this occurs, the function *diversify\_search* executes a “random walk”. The random walk consists of a number of randomly chosen moves. The number of random moves is



the minimum of either  $(1 + R_{avg} / 2)$  or the total number of possible elementary moves from the current solution. The execution time of each random step is recorded ( $\Lambda(\mu) \leftarrow t$ ) so that they become tabu. This discourages the search trajectory from returning to an old region in the search space. Note that  $\Pi$  and  $\Phi$  but not  $R_{avg}$  and  $S^{(t)}$  are cleared before the random walk is executed.

#### **function diversify\_search**

Clear the memory structures  $\Pi$  and  $\Phi$

$Q \leftarrow \{\min(1 + R_{avg} / 2, |M|)\}$  moves sampled randomly from  $M$

**for all**  $\sigma \in Q$  *(perform random walk)*

$f^{(t+1)} \leftarrow \mu(f^{(t)})$

$\Lambda(\mu) \leftarrow t$  *( $A^{(t)}$  and  $T^{(t)}$  are implicitly updated)*

$t \leftarrow t + 1$

**if**  $E(f^{(t)}) > E_b$  **then** *(update if best solution ever found)*

$E_b \leftarrow E(f^{(t)})$

$f_b \leftarrow f^{(t)}$

**Figure 4.13** The RTS *diversify\_search* Function

## **4.3 The CORA algorithm**

This section describes the CORA algorithm in more detail. The first part (section 4.3.1) gives a short overview of how the CORA algorithm combines the GNG radial basis function neural network training algorithm with the RTS combinatorial optimisation technique to build fuzzy rules. Thereafter the rationale behind the particular choice of radial basis function neural network training algorithm and combinatorial optimisation technique is discussed in sections 4.3.2 and 4.3.3, respectively.

The second part of section 4.3 explains the specific implementation details of the CORA algorithm. In particular, section 4.3.4.1 and 4.3.4.2 describe how the GNG algorithm and the RTS technique have been altered to perform fuzzy rule generation and simplification. Section 4.3.4.3 describes how the performance, or fitness, of a solution that has been found by the RTS technique is calculated. Section 4.3.4.4 discusses the method used to simplify the antecedent part of a fuzzy rule in more detail. Section 4.3.4.5 describes how the set of trained and simplified rule is further reduced by ignoring superfluous rules. Thereafter section 4.3.4.6 describes the method used to smooth the predicted output surface of the fuzzy rule set. Section 4.3.4.7 describes how fuzzy rule overlapping in the attribute space is discouraged. Finally, sections 4.3.4.8 through 4.3.4.10 discuss the computational complexity of the CORA algorithm as well as the algorithmic

enhancements that have been implemented to reduce the algorithm's computational complexity and improve training speed and capability.

### 4.3.1 Overview of the CORA Methodology

#### 4.3.1.1 Membership Function Construction Using the GNG Algorithm

The CORA algorithm constructs a set of fuzzy rules as follows. A variant of the GNG algorithm is first used to train a radial basis function neural network. This particular network structure contains one hidden layer and a single summation node in the output layer. Cell nodes with hyperspherical Gaussians are used exclusively in the hidden layer. No normalisation is performed. This has the advantage that outliers in the data do not activate any Gaussian very much and therefore have little adverse affect on the overall network performance (Fritzke, 1994b).

The radial basis function neural network is then translated into a set of purely conjunctive 0<sup>th</sup>-order Sugeno rules. Specifically, the Gaussian of a given cell node forms the antecedent part of a rule and the weight of the connection between the cell node and the output node is the rule consequent. The hidden layer connection structure that has been created by the GNG algorithm is stored for later use by the RTS algorithm. The connection structure indicates which cell nodes in the hidden layer of the trained network are topological neighbours of each other (and therefore overlap) in the attribute space of the current problem. In addition, the magnitude of attribute space overlapping is also recorded for later use (see section 4.3.4.7 for details).

#### 4.3.1.2 Rule Antecedent Assembly with the RTS Algorithm

Next, the antecedent parts of each fuzzy rule are broken up and distributed amongst a fixed, user-defined number of new fuzzy rules. There are fewer of these new fuzzy rules than were originally obtained from the radial basis function neural network. Specifically, the one-dimensional membership functions of an original fuzzy rule are distributed individually among the new set of fuzzy rules. The antecedent part of a new fuzzy rule therefore contains membership functions taken from a number of original, GNG-created fuzzy rules. This implies that the antecedent part of each new fuzzy rule can contain internal disjunction.

The RTS algorithm is then used to repeatedly exchange these membership functions among the fixed set of new fuzzy rules. The problem of finding the best partitioning of membership functions between the fixed set of fuzzy rules is modelled as a set covering problem. Specifically, one membership function from each of two rules is swapped with the other. Note that the total number of membership functions of each fuzzy rule remains constant. After each

swap of a pair of membership functions between two rules, the consequent of each fuzzy rule is determined using a linear regression technique. The aim of this iterative process is principally to find membership functions swaps that improve the accuracy of the set of rules. The RTS algorithm selectively implements swaps that improve the rule set performance, i.e. minimise the rule model fitness, with the aim of finding the best combination of membership functions amongst the fixed set of fuzzy rules. Repeated swapping continues until an acceptable level of predictive performance is attained or a user-defined maximum number of search iterations has been completed.

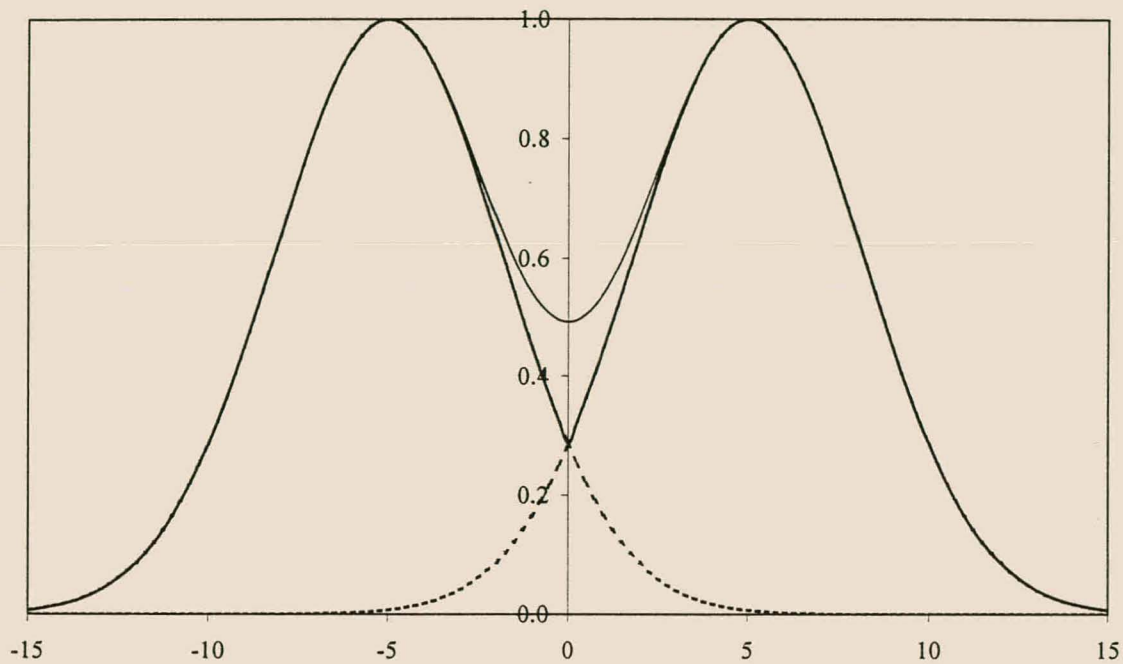
Of particular importance is that only certain membership function swaps are permitted. The choice of permissible swaps is based on the adjacency matrix information obtained from the GNG-trained radial basis function neural network. In essence, swaps are permitted only when they place a membership function in a fuzzy rule where at least one of the existing membership functions of the rule is in the direct topological neighbourhood of the candidate swap membership function. In other words, a membership function is only swapped to a rule if it is in the direct topological neighbourhood (i.e. overlaps with) of any of the membership functions currently assigned to the antecedent part of the fuzzy rule. This overlapping requirement must be valid for both fuzzy rules under consideration. The dual purpose of this swap restriction is first to bias the swapping process towards creating fuzzy rules with highly overlapping membership functions in each attribute dimension. Second, restricting swaps significantly reduces the combinatorial search space that the RTS algorithm must investigate (this latter property will be examined in more detail in chapter five).

#### **4.3.1.3 Fuzzy Conjunction and Disjunction Operators Used**

The initial set of fuzzy conjunction and disjunction operators that was used to determine the firing strength of a fuzzy rule given a training exemplar were respectively the min and max operators (Klir and Yuan, 1995; Kasabov, 1996). These will be hereafter defined as the Zadeh fuzzy operators. In the author's opinion, these Zadeh operators are the most easily understood fuzzy logic conjunction and disjunction operators. Unfortunately, preliminary experimentation showed that use of these operators produced inferior (i.e. poorer predictive accuracy) models to those obtained using other types of operators, the fuzzy operators proposed by Yu (1985) in particular. The fuzzy operators proposed by Yu will be defined as the Yu operators hereafter.

It is argued that the reason for this disparity in performance is owing to the nature of the Zadeh operators. Consider figures 4.14 and 4.15. These figures show the results of disjunction and conjunction, respectively, if either Zadeh or Yu fuzzy operators are used. Both figures show that

the Zadeh operators generate curves that are sharply peaked and discontinuous in the first derivative. In contrast, the curves generated by the Yu operators are smooth and do not exhibit peaked or spiky regions. It is hypothesised that if the CORA algorithm employs Zadeh operators, the algorithm specifically uses these spiky regions to generate improved rule models. As will be empirically shown in chapter 6, rule models constructed on this basis are capable of obtaining better training accuracy than models based on Yu operators. However, it was found that such models exhibit poorer generalisation performance on unseen data in comparison to models based on Yu operators. The CORA algorithm therefore by default employs Yu fuzzy operators.

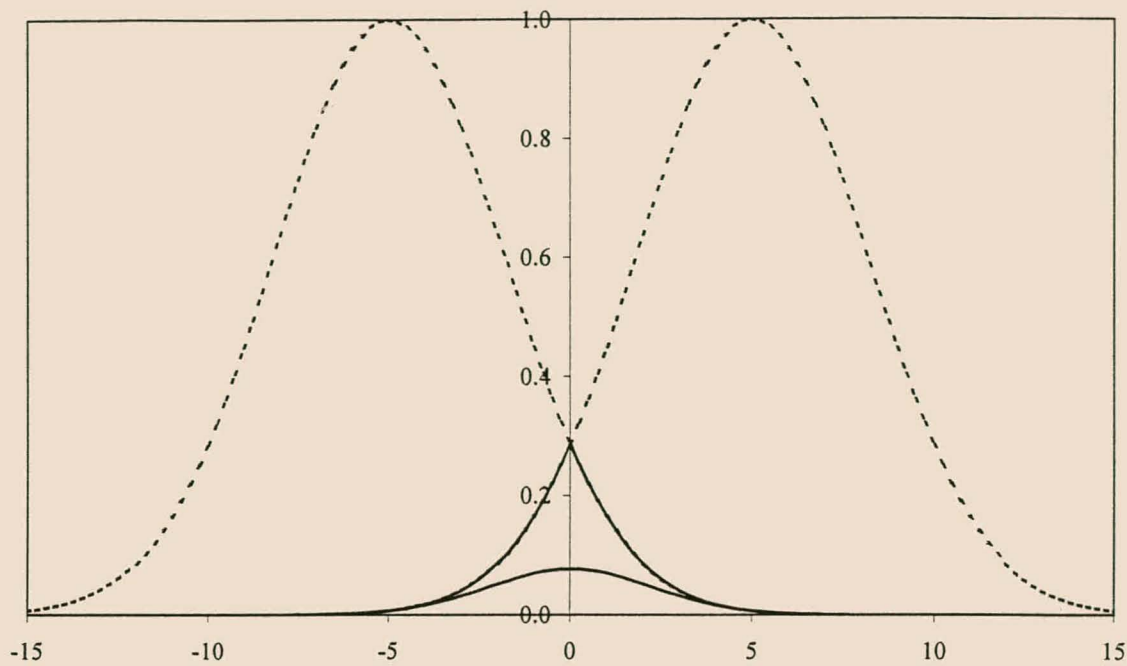


**Figure 4.14 Zadeh Disjunction Operator vs. Yu Disjunction Operator<sup>6</sup>**

---

<sup>6</sup> In each figure, two one-dimensional Gaussian membership functions are placed next to each other. These are represented by the dotted lines. The solid lines represent the results of the disjunction and conjunction operations. In figure 4.14, the top solid line is the result of the Yu disjunction operation and the bottom line of the Zadeh disjunction operation. In figure 5.15, the top solid line represents the result of the Zadeh conjunction operation and the bottom solid line represents the result of the Yu conjunction operation.





**Figure 4.15 Zadeh Conjunction Operator vs. Yu<sup>7</sup> Conjunction Operator**

#### 4.3.1.4 Membership Function Merging and Rule Reduction

The next stage of the CORA algorithm considers each fuzzy rule in turn. In each attribute dimension of the rule antecedent part, those sets of membership functions that are highly overlapping are merged with each other into single membership functions. The dual purpose of this merging exercise is first to simplify the antecedent part of each fuzzy rule. This increases the comprehensibility of the set of fuzzy rules. Second, the number of membership functions is reduced. This means that if the new set of membership functions is presented to the RTS algorithm for further membership function swapping, the size of the corresponding combinatorial problem is diminished. Such membership function merging and continued search for an optimal partitioning of membership functions would continue until an acceptably comprehensible and accurate set of rules is generated.

The final stage of the CORA algorithm performs subset selection to determine the optimal subset of the trained and merged fuzzy rules. Rules that do not significantly contribute to the training accuracy of the rule model are discarded. The number of rules that are discarded is determined

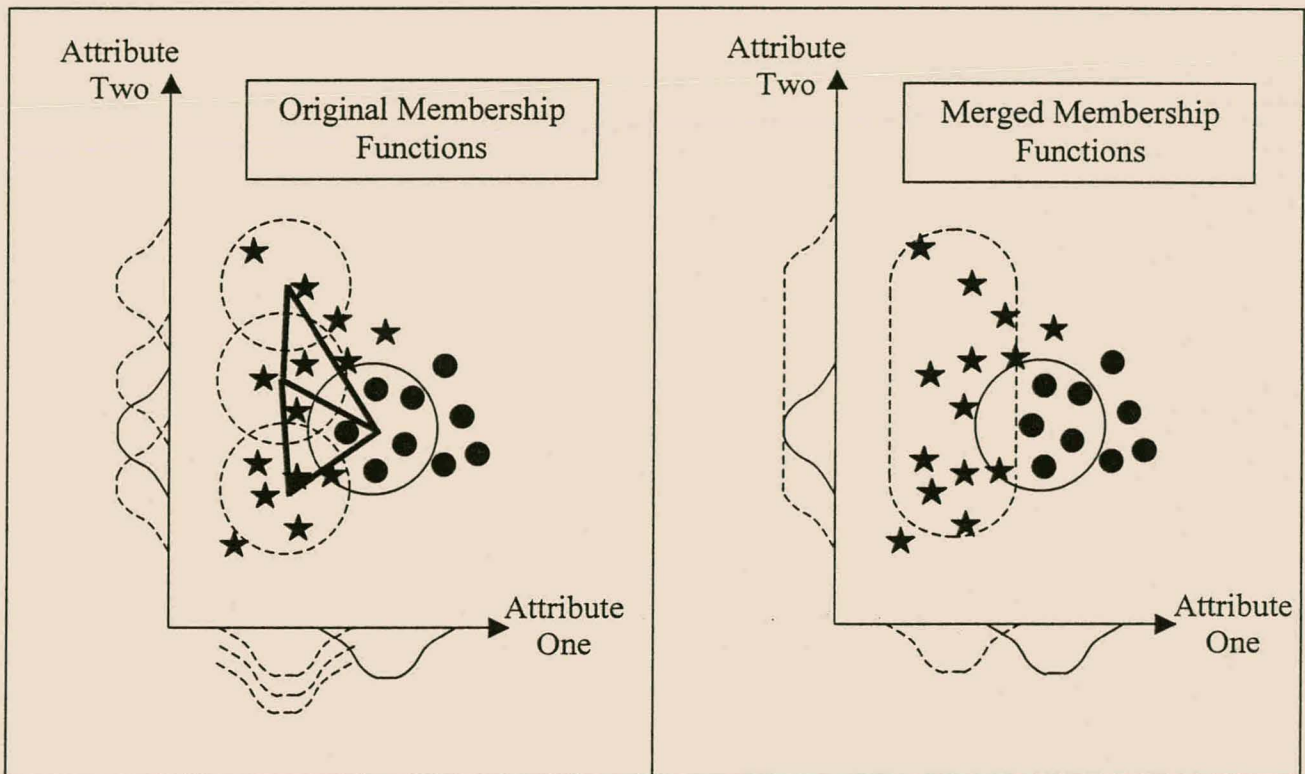
---

<sup>7</sup> The Yu conjunction and disjunction operator functions each use a single parameter. A value of  $-0.99$  is used for both functions so that the Yu operators approximate the minimum and maximum nature of the Zadeh operators as best possible without generating sharply peaked or spiky results. The value of  $-0.99$  is used throughout this dissertation.

either by the user or according to a user-defined limit on the amount of training accuracy that may be lost.

### 4.3.2 Reasons for Using the Growing Neural Gas Algorithm

The reasons for using the GNG algorithm are as follows. First, as described in section 4.1, the GNG algorithm generates a perfectly topology preserving map. In a geometrical sense, the multidimensional Gaussians of the cell nodes of the network's hidden layer will form a map that approximates the manifold formed by the training data in attribute space. This will occur even if the manifold defined by the data contains disjoint regions. In addition, each Gaussian can be viewed as representative of the data that it wins. The corresponding membership functions can therefore be viewed as building blocks with which attribute space regions that are homogeneous in the output space can be built.



**Figure 4.16 Membership Function Merging using GNG generated Radial Basis Functions**

As an illustration, examine figure 4.16. First, consider the lefthand side of the figure. The solid stars and bullets represent the data in attribute space of a two attribute, classification problem with two output classes (stars and bullets respectively). Assume that the four circles each represent a contour line of a two-dimensional Gaussian. Those Gaussians that predominantly cover stars have dotted contour lines. The Gaussian that predominantly covers solid bullets has a solid contour line. The corresponding one-dimensional membership functions of each Gaussian



are represented by the one-dimensional Gaussian functions projected on the two attribute axis lines. The three lefthand Gaussians cover a region in the output space that is homogeneous (i.e. has data with only the star class label as output). As shown on the righthand side of the figure, the membership functions of these Gaussians can be used as building blocks and merged to form a single, roughly hyperellipsoidal region in attribute space.

Second, GNG training generates a neighbourhood connection structure between cell nodes. The thick, solid lines running between the centres of each contour circle in figure 4.16 represent such neighbourhood connections. These neighbourhood connections can be used to quickly determine which Gaussians, and in turn membership functions, are in the topological neighbourhood of each other. The connections therefore indicate which Gaussians or membership functions most likely overlap in attribute space. As described in section 4.3.1.2, this connection structure is used to determine which membership function swaps are permissible.

Third, competitive Hebbian learning, used by the GNG algorithm, has been shown to generate a subgraph of the Voronoi tessellation corresponding to the current set of cell node centres (Fritzke, 1997). Voronoi tessellation is among all triangulations the best for function interpolation (Omohundro, 1990). Fuzzy rules derived from a GNG-trained radial basis function neural network should therefore possess good interpolative properties (see for example the experiments of Berlich, et al. (1996)).

Fourth, precursors and variants of the GNG algorithm have exhibited better performance in comparison to other algorithms on a number of problems. The GNG-trained radial basis function neural network with local linear mappings (Fritzke, 1995b), the GCS algorithm (Fritzke, 1994b) and the DCS-GCS algorithm (Bruske and Sommer, 1995a, 1995b) obtained significantly better accuracy than a number of other neural network architectures and training methods (e.g. multilayer perceptrons,  $k$ -nearest neighbour) on a ten-dimensional vowel recognition classification problem<sup>8</sup>.

Fifth, GNG learning can be programmed to automatically and easily change from unsupervised training to supervised training. The primary change that must be made is that the type of resource

---

<sup>8</sup> The best GCS generalisation result was 67% correct and the best DCS-GCS result was 65% correct. The best multilayer perceptron obtained 51% classification accuracy and the  $k$ -nearest neighbour technique obtained 56% accuracy. These results were reported in Bruske and Sommer (1995b). The GNG variant with local linear mappings attained a best generalisation accuracy of 66% (see Fritzke, 1995b) although this result was not consistently obtained.

that is stored by each cell node must be altered from unsupervised quantization error to supervised network output error. In addition, Gaussian width updating must take place in supervised learning when new cell nodes are inserted into the radial basis function neural network's hidden layer.

Sixth, the GNG algorithm does not suffer from the problems of greedy search or data fragmentation typical of decision tree or crisp set covering approaches (see the discussion presented in chapter 3). All the fuzzy rules generated by the GNG algorithm are continually evaluated and optimised using all the training data.

### **4.3.3 Reasons for Using the Reactive Tabu Search Technique**

The following are the reasons why the RTS technique is used instead of techniques such as the genetic algorithm. First, apart from being NP-complete (Garey and Johnson, 1979), the particular set covering problem that needs to be solved is nonlinear. An efficient search technique that can handle nonlinearities in the solution space is therefore required. The RTS technique is such a technique specifically designed for nongreedy search through nonlinear solution space.

Second, an optimisation technique is required that can efficiently manage the particular membership function swap restrictions described in section 4.3.1. It will be shown in the empirical study presented in chapter 6 that the number of permissible swaps is significantly less than the total number of possible swaps for a given fuzzy rule configuration. The typically stochastic nature of the crossover and mutation operators used by the genetic algorithm (see section 2.3.5.1) to search for an optimal solution makes it difficult for this algorithm to restrict swaps to those that are permissible. If the genetic algorithm were not forced to make permissible swaps only, some artificial cost function would need to be used to compel the genetic algorithm to build attribute space regions that are homogeneous in the output space. In the author's opinion, it would be difficult to create such a cost function. Another option would be to create heuristic feasibility operators that convert an infeasible configuration (constructed using swaps that are not permissible) into a feasible one (see e.g. Beasley and Chu, 1996). This would again be difficult, as it is hard to determine how best to move from an infeasible configuration to a feasible configuration without significantly changing the covering and accuracy characteristics displayed by the infeasible configuration. In contrast to these problems, the deterministic nature of the RTS technique makes it relatively easy to restrict membership function swaps to those that are permissible only.

Third, a number of comparative studies between tabu search variants and other combinatorial optimisation techniques have been published. Sinclair (1993) studied simulated annealing, genetic algorithms, tabu search, the Great Deluge algorithm and the Record-to-Record Travel algorithm using thirty-seven data sets obtained from experiments involving hydraulic turbine runner balancing. Sinclair found that tabu search obtained the best solutions but at the cost of long running times.

Battiti and Techiolli (1994b) compared the results obtained by simulated annealing and the RTS technique to each other using five quadratic assignment problems. RTS obtained comparable or better solutions for four of the five problems and a significantly worse solution for the fifth problem. At equivalent running times, RTS obtained significantly better results for all tasks.

Battiti and Techiolli (1995a) extensively examined the results obtained by the RTS algorithm (with the aspiration strategy employed), repeated local minima search, simulated annealing and two genetic algorithm variants on eight small and four large, randomly generated  $N$ - $K$  problems (Kaufmann and Levin, 1987). The results are summarised in table 4.1. The table gives the number of times a global optimum was obtained by the respective algorithms. For the large problems, the optimum was assumed to be the best result obtained by any algorithm. On both the small and large  $N$ - $K$  problems, the variance in solutions found varied least with RTS and most with genetic algorithms.

Optimisation Technique	Global Optima Found on Small $N$ - $K$ Problems*	Global Optima Found on Large $N$ - $K$ Problems**
Repeated Local Minima Search	256	0
Simulated Annealing	225	1
Two Genetic Algorithm Variants	46 and 9	0 and 0
RTS	251 (RTS with aspiration)	3 (RTS without aspiration)

**Table 4.1 Performance of Combinatorial Search Methods on  $N$ - $K$  Problems**

\* Each algorithm was run 32 times on each small problem (i.e. a total of 256 times)

\*\* Each algorithm was run 32 times on each large problem but only average results were reported by Battiti and Techiolli (1995a). The table therefore gives the number of times out of four that the given algorithm found the best solution.

Next, Battiti and Techiolli (1995a) studied the performance of repeated local minima search, simulated annealing, genetic algorithms, neural networks and four RTS variants on 300 randomly generated, small multiknapsack problems as well as eight large multiknapsack problems. The results are summarised in table 4.1. For the large problems, the optimum was

assumed to be the best result obtained by any algorithm. The relatively poor performance of the neural network on the small multiknapsack problems is primarily because the neural network used by Battiti and Techiolli mostly found solutions that violated problem constraints and were therefore invalid. As shown in table 4.2, RTS found the best solution six out of eight times and the neural network two out of eight times (beating RTS by a nominal 0.1-0.2%). Simulated annealing never found the best solution but obtained solutions that were on average between 0.5% and 1.0% poorer than the best solution found.

Optimisation Algorithm	Global Optima Found in Small Multiknapsack Problems*	Global Optima Found in Large Multiknapsack Problems**
Repeated Local Minima Search	178	-
Simulated Annealing	189	0
Genetic Algorithm	247	-
Neural Network	69	2
Best RTS Variant	244	6

**Table 4.2 Performance on Small and Large Multiknapsack Problems**

\* The number of times that the global optimum is found out of 300 times is given.

\*\* Each algorithm was tested eight times only on each large multiknapsack problem. The table gives the number of times out of eight that the given algorithm found the best solution. Repeated local minima search and genetic algorithms were not evaluated because these were reported by Battiti and Techiolli (1995a) to be too slow to find satisfactory solutions in acceptable computation time.

Finally, repeated local minima search, simulated annealing, genetic algorithms, neural networks and RTS were studied using small and large, strongly correlated, multiknapsack problems. See Martello and Toth (1990) for details on such problems. Table 4.3 summarises the results obtained for the strongly correlated problems. The relatively poor performance of the neural network on the small problems is again ascribed to the fact that most solutions found by the neural network violated problem constraints. On the eight large, strongly correlated problems that were considered, simulated annealing, and neural networks in particular, performed better than RTS.

In summary, the comparative studies indicate that tabu search more often finds the global optimum or the best-known solution in comparison to any of the other techniques considered. In addition, if the global optimum is not found tabu search (RTS in particular) obtains results that exhibit less variance than those of other techniques.

Optimisation Algorithm	Global Optima Found on Small, Strongly Correlated Problems*	Global Optima Found on Large, Strongly Correlated Problems**
Repeated Local Minima Search	15, 68, all	-
Simulated Annealing	4, 22, all	0 (-0.49%)
Genetic Algorithm	29, 69, all	-
Neural Network	0, 0, 16	8
RTS	15, 38, all	0 (-1.1%)

**Table 4.3    Algorithm Performance on Strongly Correlated Tasks**

- \* The three figures respectively give the number of global optima found, the number of solutions found that were within 99% of the global optimum and the number of solutions found that were within 95% of the global optimum. The experiment evaluated 100 randomly generated problems in total.
- \*\* Repeated local minima search and genetic algorithms were not evaluated because these were reported by Battiti and Techiolli (1995a) to be too slow to find satisfactory solutions in acceptable computation time. The table gives the number of times out of eight that the algorithm found the best solution. The best solution was assumed to be the best result obtained by any algorithm. The numbers in parentheses indicate the average difference between solutions found by the given algorithm and neural networks.

**4.3.4    Implementation Details of the CORA Algorithm**

**4.3.4.1    Changes Made to the GNG Algorithm**

This section describes the changes made to the GNG algorithm that is used to train the single hidden layer of the radial basis function neural network. A modification made to the learning method of the weights to the single, output summation node is also described. A change has also been made to the manner in which the “right-sized” radial basis function neural network model is found. One of the criteria that is used is the same as the one used to determine how much fuzzy rule simplification through membership function merging must take place. Consequently, a description of these criteria is postponed until the method by which the antecedent parts of fuzzy rules are simplified has been described (see section 4.3.4.4).

Both unsupervised learning and supervised learning are used one after the other to train the hidden layer of the radial basis function neural network. The purpose of such learning is to bias the hidden layer topology towards being topology preserving. A user-defined fraction of the total number of cell nodes available to the GNG algorithm is first trained using unsupervised learning, as described in section 4.1.1. The method of new node insertion has been slightly changed. The particular method used is identical to that followed by the DCS-GCS algorithm (Bruske and

Sommer, 1995b). Both the centre and resource of the new node are determined based on the proportion of resource currently allocated to the node with the maximum resource and its neighbour with the most resource. Figure 4.17 summarises the new *insert\_cell\_node* function that is used. The primary purpose of this change is to alter the network so that it reflects resource allocation after new node insertion as if the new node had been inserted from the start of training. A secondary purpose is to reduce the number of user-defined parameters required by the GNG algorithm. The changes in the way in which resource is reallocated and the new node reference vector is initialised are also used in supervised learning.

**function *insert\_cell\_node***

$q \leftarrow \arg \max r_i \text{ for all } i \in S$	<i>(determine maximum resource node)</i>
$t \leftarrow \arg \max r_i \text{ for all } i \in N_q$	<i>(<math>N_q</math> = set of direct topological neighbours of <math>q</math>)</i>
$S \leftarrow S \cup \{u\}$	<i>(create cell node <math>u</math> with centre <math>c_u</math>)</i>
$\sigma \leftarrow r_q / (r_q + r_t)$	<i>(calculate ratio of resource allocated to <math>q</math> and <math>t</math>)</i>
$c_u \leftarrow c_q + \sigma(c_q - c_t)$	<i>(calculate reference vector of new node)</i>
$r_u \leftarrow 0.5(\sigma \times r_q + r_t(1 - \sigma))$	<i>(calculate resource of new node)</i>
$A \leftarrow A \cup \{(q, u)\}$	<i>(insert neighbourhood connections)</i>
$A \leftarrow A \cup \{(t, u)\}$	
$a_{(q, u)} \leftarrow 0$	<i>(set connection ages)</i>
$a_{(t, u)} \leftarrow 0$	
Delete $(q, t)$ from $A$	<i>(remove neighbourhood connection)</i>
$r_q \leftarrow r_q(1 - 0.5\sigma)$	<i>(update resource of maximum resource node and</i>
$r_t \leftarrow r_t(1 - 0.5(1 - \sigma))$	<i>neighbour with maximum resource)</i>

**Figure 4.17 Adapted *insert\_cell\_node* Function of the GNG algorithm**

After unsupervised training has been completed, the radial basis function neural network is prepared for supervised training. The initialisation function that is used is very similar to the one described in figure 4.6, with the exceptions that no new cell nodes are created and no neighbourhood connections are formed. Rather, the existing (at least two) cell nodes and connection structure are used. In the calculation of the width of each hyperspherical Gaussian, the  $\tau$  parameter is set to one<sup>9</sup>. As described in figure 4.6, the resource of each cell node is reset to unity. The reason for this is that the network output error is used in supervised learning, instead of the quantization error. Other minor details of network initialisation are that the weights of the connections between the hidden layer and the output layer are assigned random values in the range  $[0,1]$  and that the network bias term is permanently set to zero.

<sup>9</sup> Experimentation with different values of  $\tau$  showed that the value of one resulted in the best network performance in terms of predictive accuracy and number of cell nodes used.



After initialisation the network continues to train using supervised learning (described in section 4.1.2). Two enhancements to the supervised GNG algorithm have been implemented. Specifically, if the number of training exemplars seen by the network is an integer multiple of a user-defined parameter  $\eta$ , two functions are called.

The first function recalculates the width of the Gaussian of each cell node in a fashion identical to that used during network initialisation after unsupervised training (described above). The aim of this exercise is to ensure that Gaussian widths will tend towards uniformity, rather than being very large or very small. In addition, Gaussian width updating ensures that no Gaussian permanently covers a smaller Gaussian of some other cell node. This latter phenomenon can happen if cell nodes move around significantly without new nodes being added to the network.

The second function uses a linear regression method called (amongst other names) Cholesky decomposition (Datta, 1995) to determine the optimal values for the weights to the network output layer. This regression technique will be described in more detail in section 4.3.4.3. The purpose of optimal weight calculation is threefold. The first purpose is to speed up network training by assisting the normally used delta method. The second purpose is to quickly incorporate the influence of the above-mentioned Gaussian width updating into the calculation of the outgoing weights. The third purpose is to prevent the delta method from finding weights with overly large magnitude.

#### **4.3.4.2 Implementation Details of the RTS Technique**

Three remaining issues need to be addressed with regard to the RTS technique. The first issue is the method used to initially distribute the set of membership functions obtained from the GNG-trained network amongst the fixed set of new fuzzy rules. One of two methods can be used.

The first method that can be employed clusters the original fuzzy rules obtained from the network according to the consequent (i.e. weight to the output layer) obtained for each of these fuzzy rules. There is the same number of clusters as there are new fuzzy rules. The membership functions of each cluster are then defined as the initial antecedent part of a new fuzzy rule. In other words, a new fuzzy rule is initially constructed from membership functions that belong to original fuzzy rules with roughly the same consequent. The second method that can be employed assigns the same number of membership functions to each fuzzy rule. Membership functions are randomly selected from the pool of unassigned membership functions.

The second method is used by default. The reason for this is that it was found that the first method on some problems assigned few membership functions to some rules and many

membership functions to others. Such a form of initialisation was found to generate inferior rule models in comparison to ones formed using the second initialisation method.

The second issue that needs to be addressed is the manner in which previously seen solutions are stored in memory. A hash table based on a double hashing function (Heileman, 1996) is used for this purpose. The hash key is generated using the membership function configuration of the particular solution. The following solution attributes are stored in memory. They are the solution cost function value, the last time the solution was found and the number of times the solution has been found. The membership function configuration itself is not stored in memory, owing to excessive memory requirements.

The hash key is not unique for problems that have a large number of possible different membership function configurations. For example, for a problem based on data with four attributes and using ten rules, the total number of different configurations that can be generated with the forty available membership functions is roughly  $8 \times 10^{47}$ . This number exceeds the magnitude of all integer data types used by computers available to the author. For instance, the C++ unsigned long integer data type (the CORA algorithm was programmed in the C++ language) is typically four bytes which allows unique identification of roughly  $4 \times 10^9$  configurations only. To counteract this problem, after a solution has been found in memory using the hash key, the cost function value of the stored solution is compared to that of the current solution to determine if these are the same. If this is the case, it is assumed that the two solutions are identical. See Woodruff and Zemel (1993) for more information on hashing functions for use in tabu search.

The final issue that must be discussed concerns the *check\_for\_repetitions* and the *best\_move* functions presented in figures 4.11 and 4.12, respectively. The description of the RTS technique in section 4.2 assumes that the configuration of a binary string of length  $L$  is being optimised. This is obviously not the case here. Therefore, in the *check\_for\_repetitions* function the test to determine if the repetition interval  $R$  is sufficiently short (i.e.  $R < 2(L - 1)$ ) is replaced with the test  $R \leq \text{MAX\_CYCLE}$ . MAX\_CYCLE is a user-defined constant. Furthermore the line  $S^{(t+1)} = \min(S^{(t)} \times \text{INCREASE}, L - 2)$  in the *check\_for\_repetitions* function is altered to  $S^{(t+1)} = S^{(t)} \times \text{INCREASE}$ . The first change was made to the *check\_for\_repetitions* function because it is difficult to determine a theoretical maximum cycle length for the set covering problem under investigation. The second change was made to increase the speed of the RTS technique (the RTS technique was never found to have run out of possible swaps to make).

For the same latter reason, the initial test whether  $A^{(t)}$  contains nontabu moves in the *best\_move* function, and its subsequent updating of  $S^{(t)}$  and  $t_S$ , is not implemented.

#### 4.3.4.3 Calculation of a Fuzzy Rule Consequent and Solution Fitness

This section describes the use of Cholesky decomposition (Golub and van Loan, 1993; Datta, 1995) in more detail. There are two places where the method is used. The first is during the training of the radial basis function neural network. As described in section 4.3.4.1, the delta method and Cholesky decomposition are both used to determine the outgoing weights (i.e. the consequent of each fuzzy rule) of the network. The second place where Cholesky decomposition is used is where the RTS technique determines the cost function value of a membership function configuration.

Cholesky decomposition is used to solve overdetermined, linear systems of the form  $Ax = b$ . In both cases described above,  $A$  represents the  $m \times n$  (with  $m > n$ ) firing strength matrix of the given set of  $n$  fuzzy rules.  $x$  represents the  $n \times 1$  solution vector, or in other words the set of fuzzy rule consequents. The  $m \times 1$  vector  $b$  represents the true output of the  $m$  training exemplars. Let  $C$  be the augmented matrix  $[A \ b]$ . The linear system is solved by first calculating the  $n \times n$  product matrix  $P: P = C^T C$ . Thereafter LU decomposition based on Gaussian elimination with partial pivoting is used to factorise  $P$  to determine the solution vector  $x$  as well as the estimated output  $b'$ . See Golub and van Loan (1993) as well as Datta (1995) for more details regarding these matrix operations.

In the first case mentioned above where the network outgoing weights need to be calculated, the entire set of matrix operations are performed. In addition to the above calculations, the RTS technique also calculates the residual sum of squares  $\|Ax - b\|_2^2$ , with the residuals being  $(Ax)_i - b$  for data exemplar  $i$ . This residual sum of squares is used as a component of the fitness (cost function value) of the current solution. See sections 4.3.4.6 and 4.3.4.7 for more details on how the final fitness value is determined.

It must be noted that in the calculation of the network weights or the fitness of a solution generated by the RTS technique, the set of fuzzy rules is forced to fit the complete input / output relationship. In other words, the bias term in the radial basis function neural network is permanently set to zero and no constant term is calculated by the Cholesky decomposition method. The reason for this design decision is to improve the comprehensibility of the given set of fuzzy rules and to speed up the RTS technique.

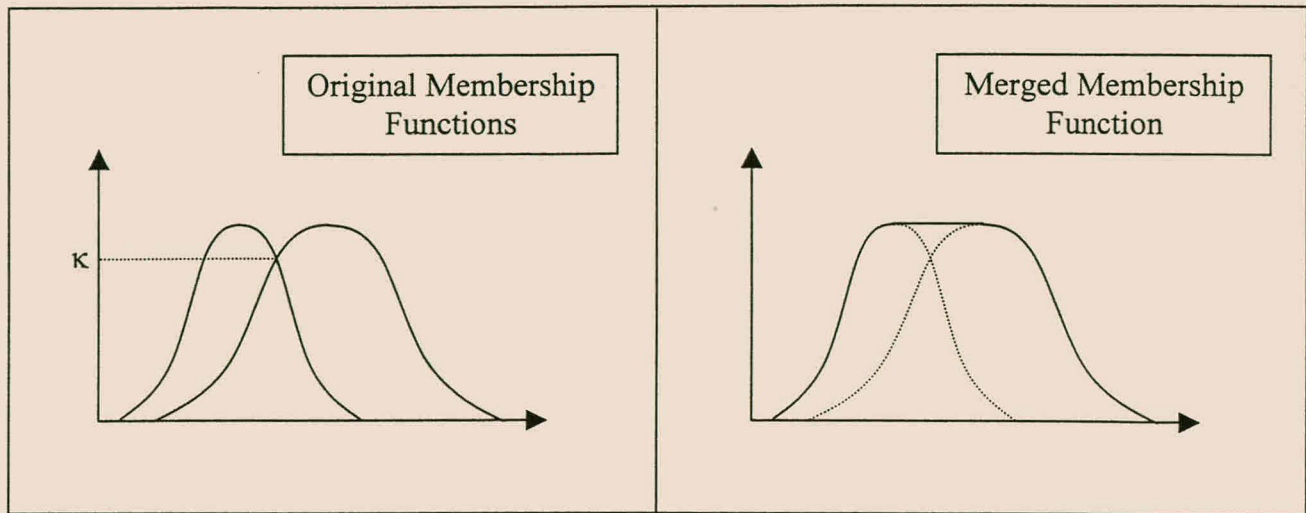
#### 4.3.4.4 The Combinatorial Rule Assembler and Obtaining the “Right-sized” Model

This section elaborates on the method used to simplify the antecedent parts of the set of fuzzy rules created using the RTS technique. Antecedent part simplification is performed once the RTS algorithm has found a solution with satisfactory accuracy or a maximum number of RTS search iterations has been completed. The criteria used to determine the best level of simplification are identical to the ones used to determine the best radial basis function neural network size. The last part of this section therefore discusses the chosen criteria in the context of both fuzzy rule simplification and network size optimisation.

A number of methods have been proposed for simplifying a set of fuzzy rules (e.g. Song, 1993; Setnes, 1995; Setnes, et al., 1998). The purpose of such methods is typically one or more of three things. The first is to simplify the antecedent part of a fuzzy rule by merging membership functions that are sufficiently close to each other into a single membership function of the same type. The second is to reduce the number of membership functions used in a set of fuzzy rules, and if possible delete copies of identical rules. This is done by redefining similar membership functions of different fuzzy rules in terms of a single, representative membership function. The third purpose is to delete membership functions that make an insignificant contribution to the fuzzy rule output.

The purpose of fuzzy rule simplification, as it is employed by the CORA algorithm, is most like, but not identical to, the first purpose described above. There is one major difference. Merging by the CORA algorithm repeatedly attempts to replace two membership functions that are sufficiently close to each other with one that best fits the profile of the original two. The dual purpose of such merging is to minimally alter the firing strength properties of the fuzzy rule antecedent part while attempting maximum simplification. The merged membership function can be of a different type to the original two. This is in contrast to methods such as that of Song (1993) which merge two membership functions into a single one of the same type. Consider the following illustration of the merging process (figure 4.18).



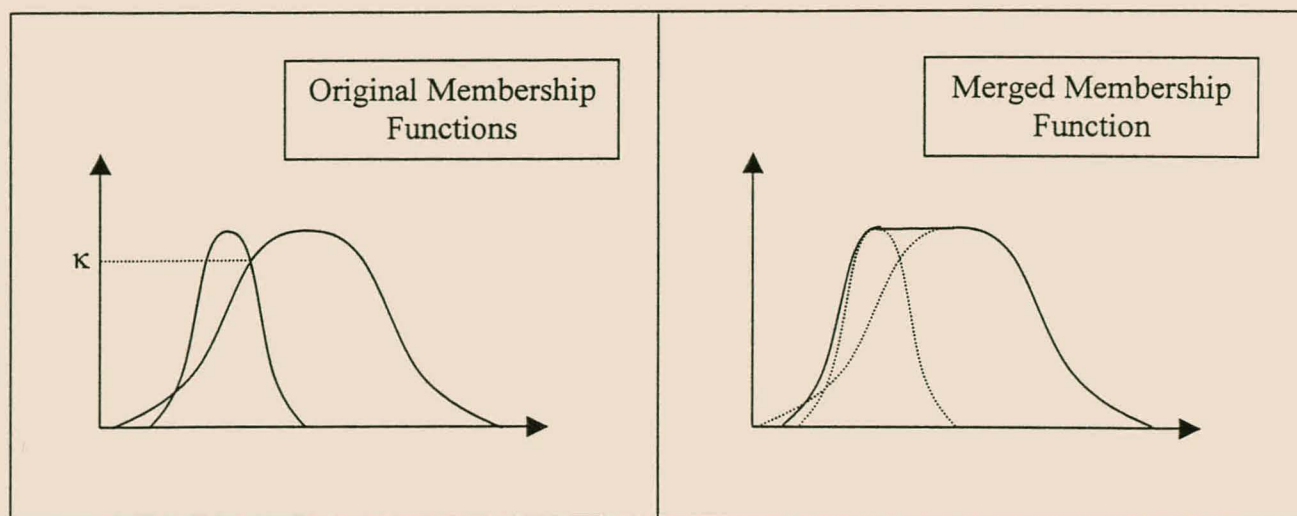


**Figure 4.18 Merging Two Membership Functions that Intersect Once**

Two one-dimensional, Gaussian membership functions are shown on the left of figure 4.18. Each represents a different linguistic term of the same linguistic variable. The two Gaussians intersect at a value of  $\kappa$ . The simplification process employed by the CORA algorithm merges the two membership functions to form a single, one-dimensional, two-sided Gaussian (shown on the righthand side of figure 4.18). The functional form of this latter Gaussian is identical to that of the two-sided Gaussian described in section 2.1.2.2. The lefthand and righthand centres of the two-sided Gaussian are defined to be the same as the centres of the original lefthand and righthand Gaussians, respectively. The two-sided Gaussian has a truth-value of one between the two centres of the function. The value of the width parameter of the lefthand shoulder is chosen to be the one that best fits the profile that is formed by the lefthand sides of both original Gaussians. In figure 4.18, it is the width of the original, lefthand Gaussian.

In some cases, such as the one illustrated in figure 4.19, the determination of a proper width value is more complicated. In figure 4.19 the lefthand shoulders of the two original Gaussians intersect. In this case the width value is chosen as the one that generates a lefthand shoulder of the two-sided Gaussian that best approximates (in the least squares sense) the maximum lefthand profile formed by the original two Gaussians. The same method is used to find the best righthand shoulder width value.

Such merging is sequentially attempted for all pairs of membership functions (original and previously merged) whose intersection point  $\kappa$  exceeds a certain threshold.



**Figure 4.19 Merging Two Membership Functions that Intersect Twice**

Owing to the nonlinear nature of the antecedent part of each fuzzy rule as well as the interaction between the antecedent parts of different fuzzy rules, it is difficult to determine the optimum value of this threshold. Therefore, the following method is followed to determine the “right-sized” model. By “right-sized” is implied the model that best approximates the true, underlying system that generated the data under examination.

The threshold at which merging is allowed to take place is sequentially lowered by small decrements from a maximum value of one to a user-defined lower limit. For a threshold value of one no merging occurs. As the threshold decreases, so the degree of merging increases. For each threshold value the antecedent parts of the *unmerged* fuzzy rules obtained from the RTS technique are merged using the method described above. Two criteria are then calculated and subsequently used to find the best level of merging and in turn the “right-sized” model<sup>10</sup>. These are Akaike’s AIC information criterion (Akaike, 1973, 1974), given by

$$AIC = \log(RSS) + 2\left(\frac{p}{m}\right) \quad \text{Equation 4.1}$$

where  $RSS$  is the residual sum of squares calculated for the particular set of merged fuzzy rules on the training data.  $p$  is the total number of parameters required by the merged set of rules (two for each symmetrical Gaussian, four for each two-sided Gaussian, one for the consequent of each

<sup>10</sup> Other criteria for determining the “right-sized” model were also investigated. These included criteria based on minimum description length (Oliver and Hand, 1994; Judd and Mees, 1995), minimum message length (Oliver and Hand, 1994), generalised prediction error (Moody, 1991, 1992) and Akaike’s final prediction error (Akaike, 1970). Preliminary experiments showed that the AIC and BIC criteria produce satisfactory results.



fuzzy rule).  $m$  is the total number of training exemplars. Akaike's BIC information criterion (Akaike, 1977) is given by the following equation with  $p$  and  $m$  defined as before

$$BIC = \log(RSS) + \frac{p \log(p)}{m}. \quad \text{Equation 4.2}$$

The level of merging that produces the minimum AIC and BIC values (the two criteria produced similar results) is chosen as the final level of fuzzy rule merging.

As described in section 4.3.1, the RTS technique is run until the training accuracy of the best solution found thus far is satisfactory or until a maximum number of search iterations have been completed. Membership function merging then takes place. Thereafter the merged set of fuzzy rules can again be presented to the RTS technique for further membership function swapping.

The AIC and BIC criteria are also used, together with a third criterion, to determine the optimal size of the radial basis function neural network. The third criterion is the performance of the radial basis function network on an unseen set of tuning data. In particular, these criteria are used to find the best number of cell nodes to place in the network's hidden layer. Although inefficient (and certainly open for improvement), a series of networks are generated, each with an increasing number of cell nodes. AIC and BIC values are calculated for each trained network. The network that firstly produces the best accuracy on the tuning data set, secondly exhibits the minimum AIC and BIC values and thirdly that is of reasonable size to produce good generalisation characteristics, is then used for further computation by the RTS technique.

#### 4.3.4.5 Fuzzy Rule Set Reduction

Rule set reduction is performed after rule antecedent assembly and merging have taken place. The Regression by Leaps and Bounds technique of Furnival and Wilson (1974) (see also Seber (1977)) is used for this purpose. The particular technique is used because it is guaranteed to find the best subset  $r$  of  $n$  fuzzy rules, given the integer  $r$ . In addition, the technique has been shown by Furnival and Wilson (1974) to amongst the fastest available.

This combinatorial search technique is used to select the best 1, 2, 3 to  $n - 1$  subsets of fuzzy rules from the assembled and merged set of  $n$  fuzzy rules. The smallest subset of fuzzy rules that retains a user-defined fraction of the training accuracy of the original  $n$  rules is then selected as the final assembled, merged and reduced set of fuzzy rules. This method of determining the optimum subset of fuzzy rules can certainly be improved. This was not done owing to time constraints.

#### 4.3.4.6 Rule Model Output Prediction Surface Smoothing

Preliminary experiments with the CORA algorithm showed that the RTS technique sometimes generated a rule model that exhibited excellent training accuracy but that generalised poorly. Inspection of these models revealed that in many cases the magnitudes of some of the fuzzy rule consequents were disproportionately large. One of the principal results of this phenomenon is that the predicted output surface produced by the rule model is excessively irregular, i.e. has large second order derivatives. In other words, the RTS algorithm has generated a model that overfits on the training data, i.e. models the training data too exactly, and because of this generalises disproportionately poorly.

A consequent magnitude penalty factor, also called a weight penalty factor, is consequently used to diminish the negative effects of the above-mentioned problem. The magnitude of this factor is calculated as the sum of the squared consequent of each fuzzy rule that makes up the current rule model. A user-defined fraction of this factor is added to the current rule model fitness. (As described in section 4.3.4.3 this initial fitness is the residual sum of squares obtained by the current rule model.) Such consequent magnitude penalisation is employed throughout the construction of the fuzzy rule model by the CORA algorithm.

The phenomenon of disproportionately large fuzzy rule consequents, or in different terms the generation of linear regression coefficients that exhibit large variances, is often the result of multicollinearity in multiple regression data. The problem of multicollinearity and ways to combat it, e.g. using ridge regression, is widely discussed in the statistical literature (see e.g. Judge (1987)). Although it is possible to use techniques such as ridge regression to combat the problems associated with multicollinearity, it was decided to use the above-described weight penalty factor method in the RTS algorithm. The reasons why the latter method is used are that the weight penalty factor method is easy to implement, is relatively fast computation-wise and gives one an explicit indication of how the RTS technique handles multicollinearity.

#### 4.3.4.7 Overlapping of Fuzzy Rules in the Attribute Space

The above-mentioned preliminary experiments also brought to light that fuzzy rules generated by the algorithm tended to overlap in the attribute space to a considerable degree. This phenomenon can be attributed to the inherent bias in membership function swapping, i.e. to create fuzzy rules with highly overlapping membership functions. In addition, the method used by the supervised GNG algorithm to determine the width of the hyperspherical Gaussians (see section 4.3.4.1 for more details) does not take attribute space overlapping explicitly into account. Furthermore, the

CORA algorithm in effect uses the weighted sum operator (Jang, et al., 1997) to determine the overall fuzzy rule set output. The result of these three factors is that the set of fuzzy rules turns into a distributed processing system. In other words, more than one rule needs to be inspected to determine the overall fuzzy output.

One way to handle this problem is to determine the overall output of the set of fuzzy rules via the weighted average defuzzification operator (Jang, et al., 1997; Lee and Park, 1997). If this operator is used there is no loss of membership function linguistic meaning. Unfortunately this operator is significantly more computationally expensive than the weighted sum operator. Normal linear regression methods such as Cholesky decomposition can also not be used to determine the fuzzy rule output and solution fitness without considerable modification.

A different method of reducing attribute space overlapping is therefore used by the CORA algorithm, in particular by the RTS technique (no changes have been made to the GNG algorithm). The method is similar to the one described in section 4.3.4.6.

In particular, for each training exemplar, the sum of the firing strengths of all the rules with respect to the specific exemplar is calculated. The least loss of membership function linguistic meaning is attained if the sum of firing strengths is at most unity (Jang, et al., 1997). Consequently if the sum of firing strengths for a particular training exemplar is greater than one, the squared difference (between one and the sum of firing strengths) is added to an initially zero penalty term. The final overlap penalty factor value is calculated over all the training exemplars.

The magnitude of the overlap penalty factor is then compared to the value obtained by the GNG algorithm using exactly the same calculation. If the value for the current rule model is greater than that of the GNG-trained rule model, a user-defined fraction of this overlap penalty factor is added to the existing fitness value of the current rule model. The effect of this form of overlap penalisation is to discourage rule models that exhibit higher attribute space overlap in comparison to that of the GNG-trained rule model.

The reasons why this particular form of overlap penalisation is used are as follows. First, preliminary experiments showed that any form of overlap penalisation reduces attribute space overlap but also starts to reduce the training accuracy of generated rule models. In the extreme, too much overlap penalisation makes it impossible for the RTS technique to build rule models with good training accuracy. Second, the above-mentioned overlap penalisation methodology is followed because it makes overlap penalisation relatively insensitive to the factor by which it is multiplied before being added to the current model fitness. In other words, the user does not have

to repeatedly tune the multiplication factor each time a rule model is constructed for a different data set. The final reason why this particular methodology is followed is that the author became aware of the attribute overlap problem at a relatively late stage of the development of the algorithm and its corresponding software implementation. Owing to time constraints, a more elegant solution to the problem of attribute space overlapping was therefore not possible.

In summary, the fitness value is therefore a linear combination of the residual sum of squares obtained by the fuzzy rule model, the weight penalty factor and the overlap penalty factor. During fuzzy rule antecedent assembly, the RTS component of the CORA algorithm tries to minimise this composite fitness value. The purpose of such fitness minimisation is to find a fuzzy rule model that exhibits good training accuracy, has small rule consequents and has at worst the same attribute space overlapping as the overlapping obtained by the GNG algorithm.

#### 4.3.4.8 Computational Complexity

This section considers the computation complexity of the CORA algorithm. Computational complexity experiments revealed that, especially for larger problems, most computation time (typically more than 90%) is taken up by two procedures used by the RTS *best\_move* function. The first is the procedure that calculates the fuzzy rule firing strengths for a given set of exemplars. The second is the procedure that is used to determine the overall fitness (residual sum of squares plus penalty terms) of a solution found by the RTS technique. The *best\_move* function itself uses the most computation time because it uses these two procedures but also because of the large number of membership function configurations that must be tested to find the next best, permissible swap. Owing to the fact that the most computation time is taken up by these two procedures, only they will be studied here.

In the worst case (i.e. if all swaps are permissible) the total number of swaps that can be made is

$$total\ swaps = \frac{gn\left(\frac{n-1}{n}\right)\left(gn\left(\frac{n-1}{n}\right)-1\right)}{2} \quad \text{Equation 4.3}$$

where  $g$  is the total number of membership functions and  $n$  is the number of fuzzy rules.  $g$  is equal to the number of data attributes times the number of Gaussians in the hidden layer of the trained radial basis function neural network. Swaps based on two membership functions from the same rule are considered illegal (the reason for the  $(n-1)/n$  factor). Therefore at worst the computational complexity of finding the best membership function swap is

$O((g^2(n-1)^2 - g(n-1))/2)$ . It will be shown in chapter 6 that the permissible swap restriction employed by the RTS technique can reduce this complexity substantially.

Next, consider the computational complexity of determining the firing strengths of  $n$  fuzzy rules for  $m$  training exemplars, each with  $d$  attributes. Assume that every rule contains the same number of membership functions in each attribute dimension. In other words, the  $g$  membership functions have been evenly distributed between the  $n$  fuzzy rules. The antecedent part of a rule therefore contains  $g/(nd)$  membership functions in each attribute dimension. If each conjunction and disjunction operation is assumed to be of  $O(1)$  complexity, the total computational complexity of presenting each training exemplar to each fuzzy rule is

$$O\left(mn\left(d\left(\frac{g}{nd} - 1\right) + (d - 1)\right)\right). \quad \text{Equation 4.4}$$

In other words, in each attribute dimension (of a fuzzy rule antecedent part)  $g/(nd) - 1$  disjunction operations must be performed. The results of these operations in each of  $d$  attribute dimensions are conjuncted using  $d - 1$  operations. These operations must occur for each training exemplar in each fuzzy rule (represented by the  $mn$  term). If equation 4.4 is simplified the computational complexity of calculating firing strengths for  $n$  fuzzy rules is  $O(mg - mn)$ .

Golub and van Loan (1993, p.248) and Datta (1995, p.361, p. 585) state that the computational complexity of using Cholesky decomposition methods for the solution of full rank, overdetermined, least squares problems is

$$O\left(\frac{mn^2}{2} + \frac{n^3}{6}\right). \quad \text{Equation 4.5}$$

The first term of equation 4.5 indicates the number of floating point operations required for calculating the product matrix. The latter term indicates the number of floating point operations required for Gaussian elimination. In summary, determination of the fitness of a set of fuzzy rules has a computational complexity of

$$O\left(m\left(g - n + \frac{n^2}{2}\right) + \frac{n^3}{6}\right). \quad \text{Equation 4.6}$$



#### 4.3.4.9 Computational Speed Enhancements to the RTS Combinatorial Search

This section discusses those speed enhancements that were made to the CORA algorithm. Owing to the fact that most computation time is spent by the RTS technique looking for more optimal solutions, speed enhancements have primarily been made to this part of the CORA algorithm. In particular, the method by which new solutions are generated, the method by which fuzzy rule firing strengths are calculated and finally the way in which the product matrix is calculated for rule consequent determination, have all been adapted.

The first enhancement that has been made is the way in which the RTS combinatorial search generates new solutions. As described in section 4.3.1.2, the RTS algorithm uses membership function swaps to generate new, candidate solutions from the current solution, with computational complexity given by equation 4.3. In order to reduce the number of candidate solutions that are evaluated during a RTS iteration, membership function swapping is combined with membership function moving. Membership function moving involves the moving of a single membership function from one rule to another. No reciprocal membership function move occurs from the latter rule to the first. A move is allowed if the move meets the requirements described in section 4.3.1.2 for permissible swaps, in particular for the membership function that is moved and the rule that it is moved to. The effect of the move on the rule that loses the membership function is ignored.

In the worst case (if all moves are permissible), the computational complexity of performing moves rather than swaps is given by equation 4.7.

$$total\ moves = \frac{g}{n}(n-1). \quad \text{Equation 4.7}$$

As before,  $g$  is the total number of membership functions contained within the fuzzy rule model and  $n$  is the number of fuzzy rules. Equation 4.7 assumes that each rule has the same number of membership functions, viz.  $g/n$  membership functions. Each of the  $g/n$  membership functions of a particular rule can be passed to all rules, with the exception of the rule to which the membership function currently belongs. Comparison of equations 4.3 and 4.7 show that the use of moves rather than swaps to generate new solutions significantly decreases the number of fuzzy rule models that need to be evaluated during a given RTS search iteration.

Currently the RTS technique can use swaps, moves or both swaps and moves as a means of creating new solutions from the existing solution. During a single iteration, either swaps or moves can be used but not simultaneously. The use of either swaps or moves can be changed if a

new RTS search iteration starts. The ratio of RTS search iterations that use swaps to those that use moves is user-defined. The current software implementation alternates between swaps and moves during the combinatorial search.

The second enhancement that has been made to the RTS algorithm to further increase the speed at which improved rule models are obtained concerns how many swaps or moves are evaluated during one search iteration. It was found that even with the permissibility restrictions placed on swaps or moves (as described in section 4.3.1.2 and above) too many swaps or moves were evaluated for a given iteration. It should be intuitively clear that all permissible swaps or moves that can be performed during a given search iteration will not always generate new rule models that have the same fitness. At the start of the combinatorial search, many of the newly generated solutions will have a better fitness than the current solution. The opposite is true towards the end of the search process. It therefore does not make sense to always try all permissible swaps or moves because at the start of the combinatorial search it should be relatively easy to find an improved rule model.

In addition, preliminary experiments brought to light that if all permissible swaps or moves are evaluated all the time the RTS technique performs relatively few search iterations in computationally acceptable time. This in turn results in the RTS reactive mechanisms (e.g. the *check\_for\_repetitions* and *escape* functions (section 4.2.4)) performing suboptimally. In particular, if only few search iterations are performed the reaction mechanisms do not quickly detect (in terms of the number of rule model evaluations) if the combinatorial search has become stuck in a search space attractor. This means that much more computation time is spent evaluating solutions in the direct search space neighbourhood of the current solution rather than moving around in the search space looking for new local optima to investigate.

The following methodology is employed to combat this problem. At the start of the combinatorial search only few of all permissible candidate solutions are evaluated during a given search iteration. The level of the so-called “search resolution” is typically set at one percent. The candidate solutions that are evaluated are chosen randomly from the entire set of candidate solutions that may be generated. The level of search resolution is maintained at this low level as long as solutions with improved fitness are discovered by the RTS technique. The search resolution is increased as soon as a better solution is not discovered. In other words, while improved rule models are being found the RTS technique does not examine the direct neighbourhood of the current solution very finely. As soon as the RTS algorithm does not find a better solution, it looks increasingly finely at the neighbourhood of the current solution.

While an improved solution is not being found, the search resolution is allowed to increase by a user-defined increment until a user-defined threshold is met. As soon as an improved solution is discovered, the search resolution is reset to the level it was assigned at the start of the combinatorial search. This increasing and resetting of the level of search resolution continues throughout the RTS combinatorial search.

#### 4.3.4.10 Computational Speed Enhancements to Rule Model Fitness Evaluation

A number of enhancements have been made to the way in which the firing strength matrix is calculated for a particular rule model. These are the following. First, the truth-value, or activation, of each training exemplar with respect to each membership function is stored in computer memory and extracted when required. Second, at the start of a RTS search iteration the firing strength matrix of the existing solution is stored in memory. Subsequent membership function swapping or moving only updates the firing strengths of those rules that have been altered. The firing strengths of rules that do not participate in a specific swap or move are not changed. (Obviously, the effects of a previous swap or move must be undone as well.) The use of this speed enhancement reduces the computational complexity of calculating the firing strength matrix from  $O(mg - mn)$  to  $O(4(mg/n - m))$ . This formula is valid if either swaps or moves are utilised to generate candidate solutions. In other words, in the worst case the firing strengths of four fuzzy rules need to be updated in comparison to the  $n$  fuzzy rules that needed to be updated before.

Finally, the intermediate results of firing strength calculations are stored in memory. If these are not affected by a membership function swap or move they can be used directly in firing strength calculations without the need for recalculation. For example, consider the calculation of the firing strength of a single fuzzy rule for a single data exemplar. The operations performed to obtain the firing strength are as follows. The disjunction of the truth-values of all the membership functions based on the first data attribute is first calculated. The result of this calculation is then conjuncted with the result of the disjunction calculation for the membership functions based on the second attribute, and so on. These intermediate disjunction results are stored in memory. If a subsequent membership function swap or move does not affect one or more of these disjunction results then one or more stored values can be used directly in the conjunction calculation.

If it is assumed that on average only half the disjunction operations need to be performed to update a particular fuzzy rule and at most four rules need to be updated, the computational complexity of calculating the firing strength matrix is further reduced to

$$O\left(4m\left(\frac{d}{2}\left(\frac{g}{4d}-1\right)+(d-1)\right)\right) \quad \text{Equation 4.8}$$

or more simply,

$$O\left(4m\left(\frac{g}{8}+\frac{d}{2}-1\right)\right). \quad \text{Equation 4.9}$$

A computational speed enhancement has also been made to the way in which fuzzy rule consequents are calculated. Inspection of the calculation of the  $n \times n$ , product matrix used by the Cholesky decomposition technique shows that not all matrix elements need to be recalculated if only a portion of the firing strength matrix has been altered. Consider a single fuzzy rule  $i$  that has been changed, i.e. the rule's set of membership functions has been altered. Only those matrix elements that store the inner product between the firing strengths of fuzzy rule  $i$  and those firing strengths all the fuzzy rules, i.e  $n$  fuzzy rules, need to be changed.

If it is assumed that at most four rules need to be updated for a particular membership function swap or move then the computational complexity of performing the Cholesky decomposition is reduced to the formula presented in equation 4.10. In equation 4.10 the  $mn^2/2$  term in equation 4.5 is reduced to  $4mn/2$  or more simply,  $2mn$ .

$$O\left(2mn + \frac{n^3}{6}\right). \quad \text{Equation 4.10}$$

Subsequently, the computational complexity of evaluating a new solution, given that the complete firing strength and product matrices of the current solution are available, is given by equation 4.11. Equation 4.11 assumes that the information for four fuzzy rules needs to be updated.

$$O\left(4m\left(\frac{g}{8}+\frac{d}{2}-1\right)+2mn+\frac{n^3}{6}\right) \quad \text{Equation 4.11}$$

# Chapter 5

## Evaluation of the CORA Technique

This chapter presents details of the experimental method used to evaluate both the performance and properties of the CORA algorithm. In particular, section 5.1 describes the issues that are addressed. These are primarily the predictive performance and complexity of the models derived by the CORA algorithm, the contribution of the different algorithmic components and the influence of training parameters.

Section 5.2 discusses the experimental method that is used to evaluate the CORA algorithm. Based on recommendations of Salzberg (1997) the evaluation of the CORA algorithm is split into two parts. Chapter 5 and 6 are principally concerned with an appraisal of the CORA algorithm itself. A comparison of the results obtained by the CORA algorithm with those results obtained by other techniques is included in this evaluation. However, it must be emphasised that this comparison is not meant to show that the CORA algorithm is able to obtain better or worse results than existing algorithms. Rather, the aim of this particular comparison is to determine whether the results generated by the CORA algorithm are competitive with those of other techniques, rule construction techniques in particular. Furthermore, most of the data sets investigated in this chapter were used during the development of the CORA algorithm itself. They were also used by the author to gain familiarity with all the other rule construction and regression techniques considered in this dissertation. It would therefore be unfair to compare the performance of the CORA algorithm to that of other techniques based on these data, with the aim of determining whether the CORA algorithm obtains significantly better results. The investigation into whether the results (e.g. predictive accuracy, model complexity) obtained by the CORA algorithm are in fact statistically different from those obtained by existing techniques



is considered in detail in chapter 7. This is the second part of the evaluation of the CORA algorithm. The data considered in chapter 7 were not used during the development phase of the CORA algorithm and will therefore allow a more unbiased estimation of the CORA algorithm's capabilities in comparison to other techniques.

Section 5.3 describes the data and algorithms considered in this investigation. Details are given on which problems were selected for study, the preprocessing performed on the data and the way that the different modelling techniques were utilised.

## 5.1 Issues that are Investigated

### 5.1.1 Predictive Performance and Model Complexity

The two most important issues that are studied in chapter 6 are the predictive performance and complexity of the rule models that are generated by the CORA algorithm. By predictive performance is meant the regression or classification accuracy of a trained rule model on unseen data, i.e. data that were not used during training. Model complexity is defined in terms of four different measures. These are:

- The number of “if...then...” rules that are used in the rule model.
- The total number of concepts used. The number of concepts of a given rule model is the total number of antecedents used by all the rules plus the number of rule consequents. As an example of how this calculation is made, consider the rule “If the reactor temperature is *Low* or *Moderate* and the feed flow rate is *Generally High* then the product quality is *Very Good*”. This rule is made up of three antecedents and one consequent. The number of concepts contained within this rule is therefore four. Note that an antecedent that uses a disjunction (OR) operator, such as the first one in the rule above, is assumed to consist of two concepts. No distinction is made between crisp and fuzzy antecedents. The consequents of a crisp rule, a Mamdani fuzzy rule or a 0<sup>th</sup>-order Sugeno fuzzy rule are each assumed to be a single concept.

For decision tree-based models, e.g. those induced by CART, the number of concepts is obtained by first rewriting the decision tree in the form of a set of rules, one rule per leaf node. The number of concepts is then determined as for “if...then...” rules.

- The total number of parameters used to define the rule model. This measure is used for fuzzy rule models only. This is because the crisp rules considered in this chapter

all consist of natural language constructs only, rather than for example membership functions that each have an underlying, parameterised, mathematical formula. All fuzzy rule models generated in this study use either Gaussian or two-sided Gaussian membership functions. As described in section 2.1.2.2, the Gaussian membership function is defined by two parameters, viz. a centre and a width. The two-sided Gaussian membership function is defined by four parameters. These are the centre and width of each of the lefthand and righthand shoulders of the membership function.

The number of parameters of a fuzzy rule consequent depends on the type of rule and the consequent membership function that is used. A Mamdani rule with a Gaussian membership function as consequent uses two parameters. A 0<sup>th</sup>-order Sugeno rule has a crisp consequent and the consequent is thus described by a single parameter. For example, if the above-mentioned fuzzy rule uses simple, one-dimensional Gaussian membership functions only, its antecedent part requires six parameters and its consequent part two parameters.

- The average number of rules that need to be evaluated to determine the final output of the rule model. For crisp decision tree or crisp rule models such as those induced by CART (Breiman, et al., 1984) or BEXA (Theron, 1994), the average number of rules that need to be evaluated is always exactly one. The reason for this is that although algorithms such as BEXA or RISE (Domingos, 1994) can construct models in which some rules overlap in the input space, only one rule is used to determine the final output of the rule model. This happens even if more than one rule fires for the data exemplar for which an output must be determined.

Four of the algorithms that are evaluated in this investigation do not generate “if...then...” rules. For these algorithms model complexity is quantitatively specified in terms of the number of model parameters only.

## 5.1.2 Influence of CORA Algorithm Training Parameters

The next part of chapter 6 examines the influence of the training parameters of the CORA algorithm. The parameters that are studied are the following (listed in order of interest):

- The number of fuzzy rules that are assembled by the CORA algorithm.
- The percentage of either swaps or moves that are evaluated during a given CORA training iteration. As described in sections 4.3.1.2 and 4.3.4.9, the CORA algorithm is able to either swap one-dimensional, Gaussian membership functions between two rules or to move a single one-dimensional, Gaussian membership function from one rule to another in order to generate a new rule model.
- Whether only swaps, only moves or a combination of both of these are used during CORA training.
- The level of consequent magnitude penalisation. As described in section 4.3.4.6, this parameter is used to keep the magnitudes of the assembled fuzzy rule consequents small and thus bias the search for the optimal rule model towards models that generate a smooth output prediction surface, rather than an irregular one.
- The level of fuzzy rule input space overlap penalisation. As described in section 4.3.4.7, this parameter influences the average number of rules that need to be evaluated to determine the final output of the trained rule model as well as the predictive performance of the derived rules.
- Whether either Yu (1985) or Zadeh (1973) fuzzy conjunction and disjunction operators are used during the determination of the firing strengths of a fuzzy rule.

The influence of these parameters is measured in terms of a number of different factors. The most important of these is the influence of different parameter settings on model predictive performance and complexity as well as the learning capability of the CORA algorithm. These issues will be evaluated for the CORA algorithm as a whole, as well as for each of the subcomponents of the algorithm. For the purposes of this investigation the CORA algorithm will be redefined in terms of three components or parts. The first part is made up of the reactive tabu search component of the algorithm and is denoted RTS. The second part consists of the merging component of the CORA algorithm and is denoted MRG. The third and final part of the

algorithm, the component that simplifies the merged rule set by deleting rules that do not significantly contribute to the final model training accuracy, is denoted RED.

As described in chapter 4, the CORA algorithm is also based on the Growing Neural Gas (GNG) algorithm. In order to provide a better perspective of the results obtained by the above-mentioned RTS, MRG and RED components, and because the GNG algorithm is a fuzzy rule construction algorithm on its own, the results of the GNG algorithm will be presented separately. In other words, the GNG algorithm will be seen as a separate fuzzy rule construction algorithm, even though the CORA algorithm uses the models constructed by the GNG algorithm to generate fuzzy rules.

Included in the evaluation of the different CORA algorithmic components is an examination of the applicability and ability of the measures used to determine the optimal level of membership function merging and rule reduction performed by the MRG and RED components.

## **5.2 Experimental Design**

### **5.2.1 Current Methodology for Comparing Algorithms and Models**

A number of researchers from both the statistical community and the artificial intelligence community have studied the experimental methodology used by artificial intelligence researchers. In particular, the methodology followed by researchers that have proposed techniques for data analysis has been examined. King, et al. (1995) found that many comparative studies have been intrasubject studies, i.e. new algorithms have only been compared to existing algorithms in the same field of research. This makes it difficult to establish the relative merits of different data analysis techniques. In addition, most comparative studies consider too few data sets to draw very general conclusions. Prechelt (1996) surveyed more than one hundred articles on neural network learning. He observed a general lack of comparative evaluation as well as the use of too few, and often artificial, data sets. For example, 29% of new algorithms were not evaluated on any real or at least realistic learning problem. Flexer (1996) studied the qualitative aspects of comparative evaluations presented in sixty-one articles from two leading neural network journals (Neural Networks and Neural Computation). He found that only 4.9% of the publications reported that a third independent data set was used for algorithm parameter tuning. In addition, only 4.9% of the publications used a statistical test to determine the relative significance of the performance results for the different algorithms that were studied.

Such lack of adequate experimentation or poor experimental methods, as observed by Prechelt and Flexer, means that it is difficult to determine whether new algorithms are in fact significantly better than existing techniques. The same problems, i.e. few comparative results and the investigation of few data sets, were found in the evaluation of fuzzy modelling publications described in section 2.3.1. In addition, as discussed in section 2.3.1, many fuzzy modelling publications present results based on data that have been repeatedly examined in the past. This again makes it difficult to ascertain whether improved algorithms are in general better than older ones, rather than just on the well-known data sets.

## 5.2.2 Experimental Method Used

In light of the above observations a modified version of the experimental method recommended by Salzberg (1997) is used to evaluate the CORA algorithm. The evaluation is split into two parts. The first part is presented in chapter 6 and the second part in chapter 7. The purpose of the former is to evaluate the CORA algorithm itself. Results comparing the rule models generated by the CORA algorithm with those derived by other techniques are used to determine whether the CORA algorithm is a competitive technique. These results are not meant to show that the CORA rule models are statistically significantly better or worse than models obtained using other techniques. Such a study is performed in chapter 7 where data are analysed that has not been used before. In other words, the aim of the investigation presented in chapter 7, in contrast to that of this chapter, is to determine whether the CORA algorithm is capable of producing models that are better than those generated by existing techniques.

The reason for this split in the evaluation of the CORA algorithm is that Salzberg (1997, p.325) suggests that a  $k$ -fold cross-validation approach be followed. For each fold the data must be divided into three distinct subsets of data, viz. a training set, a validation or tuning set and a testing set of data. The algorithm in question uses the training set to generate a model. The tuning data are used to tune the parameters of the algorithm to obtain the best possible solution. Finally, the algorithm trains on both the training and validation data and the resultant model is then tested on the testing data. Results are averaged over the  $k$  folds.

Performing  $k$ -fold cross-validation experiments for each of the different data sets, for each of the different algorithms and for each of the different algorithmic parameter settings that need to be evaluated is an extremely lengthy process. This is especially true for computationally intensive techniques such as the CORA algorithm. It was therefore decided to evaluate the CORA



algorithm using a two data set strategy in this chapter and a three set strategy in chapter 6. No form of  $k$ -fold cross-validation is performed.

In this chapter a data set is first randomised and then split into a training set consisting of 75% of the exemplars and a validation set consisting of the remaining 25% of the exemplars. This 75% / 25% split holds for all data sets unless indicated otherwise in table 5.1. The training data are used by each of the different algorithms and algorithmic variants to build models. Thereafter the generalisation performance of a given model is determined using the validation set of data.

In chapter 7 each data set is split into training, validation and testing subsets of data. Models are again generated using the training data. The validation data are then used to tune algorithmic parameters to obtain the best possible solution. Thereafter each algorithm constructs a model using the training and validation data combined. Finally, the predictive performance, etc. of these models is then determined in a once-off fashion on the test set of data. More detail on the experimental procedure is given in chapter 7.

In chapter 7 the results obtained by the various algorithms on the test data are compared more strictly with each other. The aim of this comparison is to determine whether the performance of the models generated by the CORA algorithm is statistically better than that of other types of models. The description of the statistical measure of significance used in these comparisons is postponed until chapter 7 because they are not used until that chapter.

### 5.2.3 Predictive Performance Estimation

Two measures are used to estimate predictive performance. For classification problems predictive performance, or accuracy, is defined as the percentage of exemplars that a trained model classifies correctly. For regression problems accuracy is defined as the root mean squared error. This is one minus the variance explained by the model over the total output variance, or in equation form:

$$R^2 = 1 - \frac{SSE}{SST}$$

$$\text{where } SSE = \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

$$\text{and } SST = \sum_{i=1}^n (y_i^2) - \frac{1}{n} \left( \sum_{i=1}^n y_i \right)^2$$

**Equation 5.1**

## 5.3 Problems and Algorithms Evaluated

### 5.3.1 Data Sets Investigated

A list of the primary characteristics of the data<sup>1</sup> studied in this chapter is presented in table 5.1. In addition, sections 5.3.1.2 through 5.3.1.11 give a short summary of the ten problems that are studied. The second column of table 5.1 denotes whether the problem is a regression problem (R) or a classification problem<sup>2</sup> (C). For classification problems the figure in brackets indicates the percentage of the data exemplars that belong to the majority class. The training data and validation data columns give the number of data exemplars used for training and validation, respectively. For the Spiral data set only training data are available. The fifth column indicates the number of problem attributes and the last column gives the percentage of these that are numeric. Numbers in parentheses indicate the actual number of attributes presented to the different training algorithms. This property of some of the data sets is explained in more detail later.

Data Set Name	Problem Type	Number of Training Data	Number of Validation Data	Input Dimensionality	Numeric Attributes (%)
Abalone	R	2350	783	8 (9)	88 (89)
Auto	R	299	99	7	100
Housing	R	407	136	13	100
Ionosphere	C (64.1)	263	88	34 (33)	100
Servo	R	125	42	4 (12)	50 (83)
Sincos	R	300	1200	4	100
Slugflow	C (62.4)	133	43	4	100
Spiral	C (50.0)	192	-	2	100
Pima	C (65.1)	576	192	8	100
WBC	C (65.5)	524	175	9	100

**Table 5.1 Details of the Data Sets Studied in Chapter 6**

#### 5.3.1.1 Reasons for Choice of Data Sets

The criteria that were used to decide which data sets to study are as follows. First, real-world regression and classification problems were preferred over synthetic problems. Unfortunately

<sup>1</sup> All the data sets excluding those for the Sincos and Slugflow problems were obtained from the UCI Repository of machine learning databases (Blake, et al., 1998).

<sup>2</sup> All classification problems have two possible classes or outcomes.

relatively few suitable regression data sets are available to the author. In contrast, there are many classification problems available at repositories such as the UCI Repository of machine learning databases (Blake, et al., 1998). Therefore although the CORA algorithm is primarily designed to solve regression problems, half of the problems examined are classification problems. This is done so that the properties of the CORA algorithm and the rule models that it derives can be evaluated over a wider range of problems.

Second, only classification problems with two output classes were considered. The reason for this is that training and validation of the CORA algorithm becomes lengthy and cumbersome for classification problems with more than two classes. This is because the GNG algorithm and the CORA algorithm have been designed for single output problems. Classification problems with two or more classes therefore have to be reworked into three or more separate two-class problems. Rule models have to be generated for each of these data sets. The results obtained by all the rule models then have to be assembled to determine the final results of the CORA algorithm on the multiclass problem.

Third, problems with varying data set sizes and input dimensionality were selected. This allows the performance of the CORA algorithm to be evaluated over a wide range of different problems. In addition, data sets with little or no missing values were chosen because the CORA algorithm has not been explicitly designed to be able to handle missing data.

Fourth, data that consist exclusively of or that have a high proportion of numeric attributes were chosen over problems that have many discrete, unordered attributes. The reason for this is that membership functions are by nature ordered and therefore it does not make sense to try to build membership functions for unordered attributes. This is especially true in the context of the CORA algorithm, for two principal reasons. First, the RTS combinatorial search is explicitly biased towards constructing rules that each has an antecedent part comprising of highly overlapping membership functions. Membership functions based on unordered attributes cannot overlap. For example, consider an unordered attribute with possible values “Apple”, “Pear” and “Orange”. A membership function could be defined for each attribute value but the numerical overlap of any two of these membership functions would not make sense. Second, membership function merging (performed by the MRG component of the CORA algorithm) is based on the principle of numerically overlapping membership functions. Membership functions that do not overlap cannot be merged.

### 5.3.1.2 The Abalone Problem

The purpose of the Abalone problem (Waugh, 1995) is to predict the age of abalone from eight physical measurements of such creatures. These measurements are the sex, maximum shell length, shell diameter, height, whole weight (weight of the entire abalone), shucked weight (weight of the meat alone), viscera weight (gut weight) and shell weight. All these measurements are continuous except for the first. The sex of the abalone can be male, female or infant. This attribute was encoded using two new attributes and the binary encoding 00, 01 and 10.

Only the 3133 exemplars normally used by researchers studying this problem for training purposes are used to evaluate the performance of the CORA algorithm. 75% of this subset of exemplars are used for training and the remaining exemplars are used for validation purposes.

### 5.3.1.3 The Automobile Miles Per Gallon Data Set (Quinlan, 1993b)

This problem is hereafter defined as the Auto problem. The aim of this problem is to predict the fuel consumption of a variety of different automobiles. The attributes of the problem are the number of cylinders, displacement and power of the engine, the weight and acceleration of the automobile, the origin of the car and finally the year in which the particular car model was first manufactured. All attributes are assumed to be numeric. The original data set obtained from UCI Repository of machine learning databases contained an eighth attribute, the name of the car. This attribute is ignored in this study. The power attribute has six missing values. These were replaced by the mean value of the power attribute before the data were examined further.

### 5.3.1.4 The Boston Housing Problem

The aim of the Boston housing problem, defined as the Housing problem, is to predict the median value of owner-occupied homes in Boston, Massachusetts, based on thirteen continuous attributes. The attributes are the per capita crime rate by town, the proportion of residential land zoned for lots over 25000 ft<sup>2</sup> (2323 m<sup>2</sup>), the proportion of nonretail business acres per town, whether the tract of land upon which the home is situated borders the river, the nitric oxides concentration in the air, the average number of rooms per dwelling, the proportion of owner-occupied houses built prior to 1940, the weighted distances to five Boston employment centres, an index of accessibility to radial highways, the full-value property tax per \$10000, the pupil to teacher ratio by town, the result of the formula  $1000(\text{Bk} - 0.63)^2$  where Bk is the proportion of black people by town and finally the percentage lower status of the population. Refer to Harrison and Rubinfeld (1978) for more details regarding this set of data.

### 5.3.1.5 The Ionosphere Data Set

This data set involves the classification of radar returns from the ionosphere. These radar data were collected by a system consisting of a phased array of sixteen high-frequency antennas. The targets were free electrons in the ionosphere. The radar returns are classified as “Good” if they show evidence of some type of structure in the ionosphere and otherwise “Bad”.

The radar data have been preprocessed using an autocorrelation function whose arguments are the time of a pulse and the pulse number. There are seventeen pulse numbers. Data exemplars are described by two continuous attributes per pulse number, corresponding to the complex values returned by the function resulting from the complex electromagnetic signal. This gives a total of 34 continuous attributes. The second of these attributes is ignored in this investigation because the values of the attribute are all identical. Refer to Sigillito, et al. (1989) for more details on this data set.

### 5.3.1.6 The Servo Problem

These data were obtained from the simulation of a servo system (Quinlan, 1992). The aim of the problem is to predict the time required for the system to respond to a step change in position set point. The first two of the four attributes of the problem are discrete. Each of these has five possible discrete values. These two attributes are encoded using ten new attributes and binary encoding. A particular set of five attributes therefore encodes one of the original, discrete attributes. A total of twelve attributes, ten binary, two continuous, were therefore presented to each of the algorithms studied in this chapter.

### 5.3.1.7 The Sincos Problem

The Sincos problem is a synthetic problem created by Schmitz, et al. (1999). Three of the attributes ( $\theta$ ,  $x$ ,  $s$ ) are continuous and the fourth ( $\phi$ ) can take on one of two numeric values. The single, continuous output is given by

$$y = \sin(4\pi\theta - \phi) + ax + bs + c\varepsilon, \quad \text{Equation 5.2}$$

where  $\theta$ ,  $x$  and  $s$  can range between zero and one.  $\phi$  can be either zero or  $\frac{1}{2}\pi$ .  $\varepsilon$  has randomly generated, normally distributed values with a mean of zero and standard deviation of one. The data used in this investigation were generated using values of 0.3, 0 and 0.2 for  $a$ ,  $b$  and  $c$ , respectively. Note that even though  $b$  is 0 and subsequently  $s$  plays no role in determining  $y$ , the various algorithms still have to deal with what is basically a nuisance variable. The validation set of data was generated with  $c$  set to 0.



### **5.3.1.8 The Slug Flow Data Set**

This data set was reported by Reimann, et al. (1992) who studied the transitions between flow patterns of multiphase fluid flow systems in horizontal pipes. Two primary flow regimes are possible, viz. slug flow or non-slug flow. (The latter regime includes transitional phases such as slug-annular and slug-wave flow.) The aim of this data set is to predict which of these two flow regimes (slug or non-slug flow) are present in the pipe from four attributes of the flow system. The attributes are the flow pressure, the diameter of the pipe, the superficial liquid velocity and the superficial gas velocity.

### **5.3.1.9 The Spiral Problem**

The Spiral data (Lang, et al., 1988) represent two distinct, two-dimensional, intertwined spirals. The data of each spiral represent a class or discrete output value. The task is to train on all the data until the algorithm predicts the correct classification for all the exemplars.

### **5.3.1.10 The Pima Diabetes Problem**

These data, defined as the Pima data set, are a subset of a larger database of results obtained from a study of the incidence of diabetes in Pima Indians (Smith, et al., 1988). The aim of the exercise is to predict whether the female patients that are at least 21 years old show signs of diabetes mellitus or not. The eight attributes available for this prediction are the number of times the patient has been pregnant, the plasma glucose concentration in an oral glucose tolerance test, the diastolic blood pressure, the triceps skin fold thickness, the two-hour serum insulin concentration, a body mass index, a diabetes pedigree function and finally the age of the patient.

### **5.3.1.11 The Wisconsin Breast Cancer Problem**

The aim of this data set (Mangasarian and Wolberg, 1990; Mangasarian, et al., 1990; Bennett and Mangasarian, 1992) is to predict whether a tissue sample is benign or malignant, based on the following sample attributes. The attributes are the clump thickness, uniformity of cell size, uniformity of cell shape, marginal adhesion, single epithelial cell size, the number of bare nuclei, the bland chromatin, the number of normal nucleoli and finally the level of mitoses. The original data set obtained from the UCI repository contains a tenth attribute, the sample code number. This attribute is ignored in this investigation. Sixteen exemplars contained a single missing value each. These were assumed to be the mean of the given attribute.

This data set is referred to as the WBC problem, both in table 5.1 and the rest of this dissertation.

## 5.3.2 Algorithms Evaluated in Chapters 6 and 7

### 5.3.2.1 Crisp Rule Construction Algorithms

Apart from the rule models derived by the CORA algorithm, the results generated by the following algorithms are also evaluated both in chapter 6 as well as chapter 7. For classification problems CART (Breiman, et al., 1984) and BEXA (Theron, 1994) are studied. CART is also used for regression problems. The logical tests of the CART decision tree nodes are restricted to be functions of a single attribute. In other words, linear combinations of attributes are not permitted. Both these algorithms are described in chapter 2 of this dissertation. These are the only crisp decision tree or rule construction algorithms considered. The reasons why these two algorithms are used are as follows. First, software implementations of these algorithms are available to the author. Second, CART is typically used as a benchmark decision tree algorithm. Third, as discussed in section 2.2.2, BEXA is considered to be a state-of-the-art, crisp rule induction algorithm. The results obtained by BEXA should therefore give a good indication of the capabilities of advanced, crisp rule construction methods.

### 5.3.2.2 Fuzzy Rule Construction Algorithms

The rule models generated by two techniques that construct fuzzy rules are also studied. Both techniques build radial basis function networks (described in section 2.3.6.1) with one hidden layer and a single node in the output layer. The hidden layer kernel functions are multidimensional, spherical Gaussians. The output node is a linear summation node. The first technique trains the hidden layer using the Growing Neural Gas (GNG) method (Fritzke, 1994a) and the second technique uses  $k$ -means clustering (Moody and Darken, 1989). The weights to the output layer of the GNG network are trained using a combination of the delta rule (Widrow and Hoff, 1960; Stone, 1986) and multiple linear regression (Seber, 1977). The weights to the output layer of the  $k$ -means, radial basis function network are trained using linear regression only. These networks are applied to both classification and regression problems.

The  $k$ -means, radial basis function network has a bias and the input nodes are also directly joined to the output layer by weighted connections. This means that the network function contains both linear terms and a constant in addition to the nonlinear terms represented by the network's hidden layer. The GNG network has neither a bias nor any linear terms. The GNG network is therefore directly equivalent to a set of 0<sup>th</sup>-order Sugeno fuzzy rules. The  $k$ -means, radial basis function network also generates a set of 0<sup>th</sup>-order Sugeno fuzzy rules but these are combined with a constant fuzzy rule (the bias term) and a linear function (the set of linear terms).

The reasons why these two techniques are evaluated are because both construct 0<sup>th</sup>-order Sugeno fuzzy rules comparable to those generated by the CORA algorithm and software implementations of these techniques are available to the author.

### 5.3.2.3 Other Regression Techniques

In addition to the aforementioned rule construction techniques the results obtained by two variants of a multilayer perceptron trained with the generalised delta rule, multiple linear regression and the MARS algorithm are also studied. These techniques do not construct rule-based models. The reasons why they are considered are as follows. Multiple linear regression is used because it is a very well known technique and therefore provides a good baseline against which other regression techniques can be compared. The two multilayer perceptrons are considered because such networks are often used as a benchmark in the evaluation of new neural network architectures and training methods. MARS is evaluated because the author sees it as representative of the state-of-the-art in statistical, nonparametric modelling of regression problems.

Both multilayer perceptrons considered in chapters 6 and 7 are neural networks that each consists of a single hidden layer and an output layer with a single output node. Both variants use tanh transfer functions ( $f(x) = \tanh(\frac{1}{2}x)$ ) for the nodes of the network's hidden layer. The first variant has a summation output node. The second multilayer perceptron variant uses a tanh output node. Both neural network architectures have a bias term. The particular training method used to optimised the parameters of these networks is the generalised delta rule (Zurada, 1992; Haykin, 1994) with an adaptive learning rate.

The regression model derived by the multivariate adaptive regression splines (MARS) algorithm (Friedman, 1991) consists of a set of linear regression splines (Schultz, 1969) and products thereof. These regression splines combine to form a piecewise approximation of the output. Splines are sequentially added to the existing regression model in order to minimise a predefined error function. The algorithm finds the regions in which each spline is applicable by adjusting a set of so-called knots. After an initial growth phase, MARS prunes the model by deleting some of the splines, based on another error function. Refer to Friedman (1991) for more details regarding the training method of MARS as well as the kind of models that are formed.

### 5.3.3 Details of Experimental Procedure

#### 5.3.3.1 Data Preprocessing

Before the data sets examined in this chapter were split into training and validation sets each complete data set was randomised and then normalised to have a mean of zero and a standard deviation of one. This normalisation was done individually for all the attributes as well as the output of each problem. There are two reasons why this normalisation step was performed. The first is that both the GNG algorithm and the CORA algorithm assume that no bias term is used. In other words, these algorithms force the regression function that is constructed (the set of fuzzy rules can also be interpreted as a function consisting of a linear combination of nonlinear functions) to have an output of zero when no rule fires. Second, the data are scaled because the GNG algorithm uses the Euclidean metric to determine the distance between a hidden layer, multidimensional Gaussian and the input of data exemplars. If the data were not scaled an attribute with a high standard deviation would be seen as disproportionately more important than an attribute with a smaller standard deviation.

As described in sections 5.3.1.2 and 5.3.1.6 respectively, the discrete attributes of the Abalone and Servo data sets were binary encoded before being studied. This encoded form of the data was presented to all the algorithms considered in this investigation. This was the case even for those algorithms such as CART that are able to handle discrete attributes without the need for some form of encoding.

#### 5.3.3.2 Use of Algorithms and Parameter Settings

The techniques that are evaluated in chapter 6 were utilised in such a manner that, in the author's opinion, would maximally benefit each of the techniques, given the constraints set by algorithm software implementations as well as the time constraints of this investigation. For example, CART was allowed to use the validation set of data to choose the optimal tree during tree induction. In other words, the CART algorithm could "see" the validation data during training. Owing to the software implementation constraints of the other techniques and algorithms, in particular the lack of the capability to use the validation data as the CART software does, these algorithms were initially run a number of times until a good set of default parameters were found. Thereafter, for a particular set of data, the problem-specific parameters, such as `number_of_rbf`s (number of radial basis functions) or `number_of_rules`, were tuned based on the validation accuracy obtained by the derived models. Finally, for each nondeterministic algorithm three runs with different random seeds were performed using these optimal parameters.

The CORA algorithm was evaluated slightly differently. First, default parameters were chosen based on experience obtained during the development of the CORA software. Thereafter parameters that have the most effect on the performance of CORA algorithm were selected for detailed study. These parameters are those listed in section 5.1.2. For each parameter setting that was evaluated three different runs were performed, each with a different random seed. The GNG results (adjacency matrix, etc.) used for each of these runs correspond to the three runs with different random seeds performed by the GNG algorithm.

Three different settings were examined for each of the first three training parameters listed in section 5.1.2, viz. the number of fuzzy rules assembled, the percentage of swaps or moves evaluated for a given iteration and whether moves or swaps or both are utilised. For the consequent magnitude penalisation parameter and the input space overlap penalisation parameter only two settings were evaluated, in particular whether the given form of penalisation is used or not. If a form of penalisation is used, the penalisation factor was chosen based on a few shortened, preliminary runs that were performed to gain an idea of what the effect of the penalisation factor would be for longer runs.

The last parameter setting, whether Yu or Zadeh fuzzy conjunction and disjunction operators are used, was evaluated for only one of the data sets (the Auto problem) and for only a few of the other parameter setting combinations. The reason for this is because of time constraints placed on the experimentation that could be performed for this chapter.

Finally, note that for some problems not all combinations of training parameter settings were evaluated owing to time constraints as well as the inability to perform such experiments. For instance, for the Abalone problem the effect of the number of rules on CORA model predictive accuracy was only evaluated for a move or swap threshold<sup>3</sup> of 0.9 because it would take too long in terms of computation time to evaluate other possibilities.

---

<sup>3</sup> See section 4.3.4.9 for more details regarding this threshold.



# Chapter 6

## Results and Discussion of the Empirical Evaluation of the CORA Algorithm

This chapter presents and discusses the results obtained for the experiments performed to evaluate the performance and properties of the CORA algorithm and its three subcomponents. These subcomponents are the Reactive Tabu Search (RTS) component, the membership function merging (MRG) component and the fuzzy rule reduction (RED) component. Section 6.1 compares the models generated by the CORA algorithm with those obtained using other techniques, specifically those described in section 5.3.2. Section 6.2 examines the influence of CORA training parameters in more detail. Thereafter section 6.3 discusses miscellaneous aspects of the CORA algorithm and gives an in depth appraisal of some of the CORA algorithm and CORA rule model properties. Finally, section 6.4 presents conclusions regarding the performance of the CORA algorithm as well as the algorithm's properties.

### 6.1 CORA Fuzzy Rule Models vs. Models Derived by Other Algorithms

#### 6.1.1 CORA Algorithm Training Capability

The first aspect of the CORA algorithm that will be discussed is the algorithm's training capability. The aim of the experiments performed for this section was to determine whether the CORA algorithm is capable of retaining both the accuracy performance and geometrical

properties of the fuzzy rule model generated by the GNG algorithm. Model complexity is exclusively compared in terms of the number of fuzzy rules that comprise the final trained model. A single data set, the Spiral data set, is examined. The reason for this is that the Spiral data set is two-dimensional and has a distinctive geometrical form (two intertwined spirals). This allows one to easily gain a visual perspective of the model generated by an algorithm. In addition, the Spiral problem is often used by researchers (e.g. Carpenter, et al., 1992; Simpson, 1992; Fritzke, 1994b; Bruske and Sommer, 1995b; Domingos, 1996c; Lovell and Bradley, 1996) to determine whether a given technique is capable of solving a classification problem with highly nonlinear decision boundaries.

The experiments<sup>1</sup> performed for this section consisted of trying different GNG network sizes, i.e. different numbers of hidden layer cell nodes (or in other words fuzzy rules) until first the GNG algorithm and second the CORA algorithm obtained models with maximum classification performance. Both the GNG algorithm and the CORA algorithm were able to construct models with 100% classification accuracy on the training data (as described in section 5.3.1, the Spiral data set consists of training data only). Therefore, for this particular problem, the rule model generated by the CORA algorithm, given the correct set of training parameters, was able to retain the classification performance obtained by the GNG-trained fuzzy rule model upon which it was based.

Next consider the geometrical properties of the generated fuzzy rule models. Figures 6.1 and 6.2 present a graphical depiction of the output decision boundaries generated by the models obtained

---

<sup>1</sup> The final GNG algorithm rule model was obtained using the default training parameters presented in Appendix A, with the exception of the number of training epochs, which was set to 1090, and the maximum number of cell nodes, which was set to 105. The random seed for all GNG runs and CORA runs was set to 5.

All CORA algorithm runs were based on the optimal (100% classification accuracy) GNG model. Using this model, different CORA training parameter settings were tested until the CORA-generated rule model obtained 100% classification accuracy. The optimal CORA-generated model was obtained using the default settings described in Appendix A, with the exception of the number of training iterations (set to 5000), the number of assembled fuzzy rules (set to 30), the attribute space overlap penalty factor (set to 0.1), the consequent magnitude penalty factor (set to 0.001) and the minimum swap threshold (set to 0.01).

The principal training parameters that affected classification performance were the number of cell nodes used by the GNG algorithm and the number of fuzzy rules assembled by the CORA algorithm. For the best training parameters settings the CORA-generated rule model exhibited an average attribute space overlap of 4.56. The GNG-generated rule model had a (worse) overlap of 5.07.

by the GNG and the CORA algorithms. For the latter algorithm membership function merging and subsequent rule reduction did not change the trained fuzzy rule model in any way. In other words, each of the three CORA components (denoted RTS, MRG and RED hereafter) returned exactly the same model.

Each figure<sup>2</sup> presents a two-dimensional plot of the training data (represented by the small black circles) as well as the predicted rule model output surface. Each spiral represents a different output class. Lighter regions in the figure indicate that one class is predicted, whereas darker regions indicate that the other of the two possible classes is predicted by the given rule model.

As expected<sup>3</sup>, and as can be seen in figure 6.1, the estimated output of the GNG-derived rule model clearly forms two intertwined spirals, one lighter and the other darker (corresponding to the two different classes). This is especially true towards the centre of the figure where the training data are packed more closely together. Towards the outside of the spirals where the data become sparser the predicted output becomes less smooth, even though each training data point is correctly classified.

As indicated in figure 6.2, the predicted output of the CORA-generated rule model also forms two intertwined spirals, one lighter and the other darker. The predicted output surface presented in figure 6.2 differs primarily in two respects from the surface shown in figure 6.1. First, the spirals in figure 6.2 are more square. This is owing to the fact that the CORA-generated rule model consists of 30 rules only, in comparison to the 105 rules of the GNG-generated model. Because relatively fewer rules could be assembled, the CORA algorithm was not able to generate decision surfaces as smoothly curved as those obtained by the GNG rule model. In addition, the CORA algorithm typically generated fuzzy rules with antecedent parts consisting of a disjunction of two or more axis-parallel hyperellipsoids. Figure 6.3 depicts the firing strength (output) surface of one such fuzzy rule. The fuzzy rule shown in this figure has an antecedent part consisting of a disjunction of four hyperellipsoids. In the author's opinion it is difficult to use such hyperellipsoids to create a continuously curved, decision surface.

---

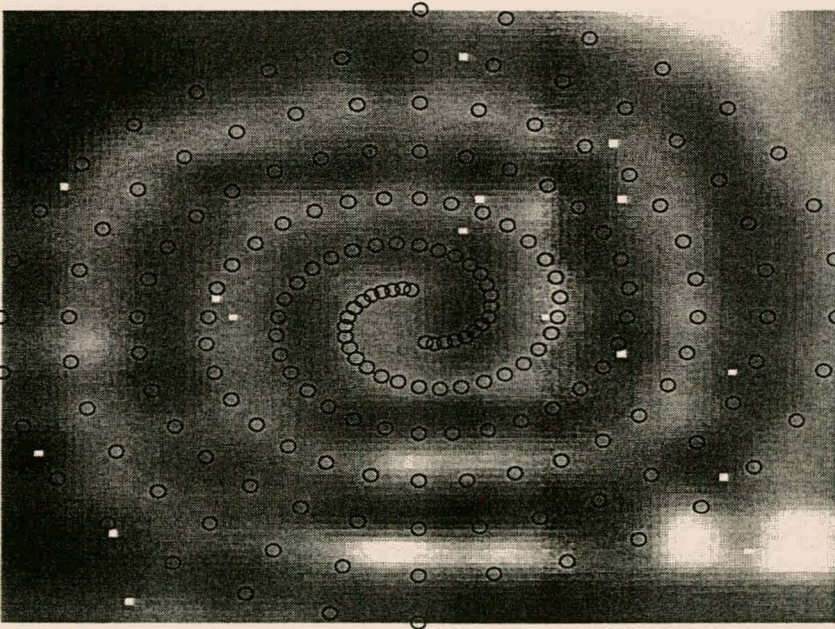
<sup>2</sup> The few small, randomly placed, white squares should be ignored. These squares are a result of the inability of the software used by the author to correctly copy pictures and have nothing to do with predicted model output or any other aspect of the derived fuzzy rule models.

<sup>3</sup> Fritzke (1994) as well as Bruske and Sommer (1995) have applied variants of the GNG algorithm on the same Spiral problem with similar success, viz. 100% classification accuracy and good representation of the double spiral geometry.



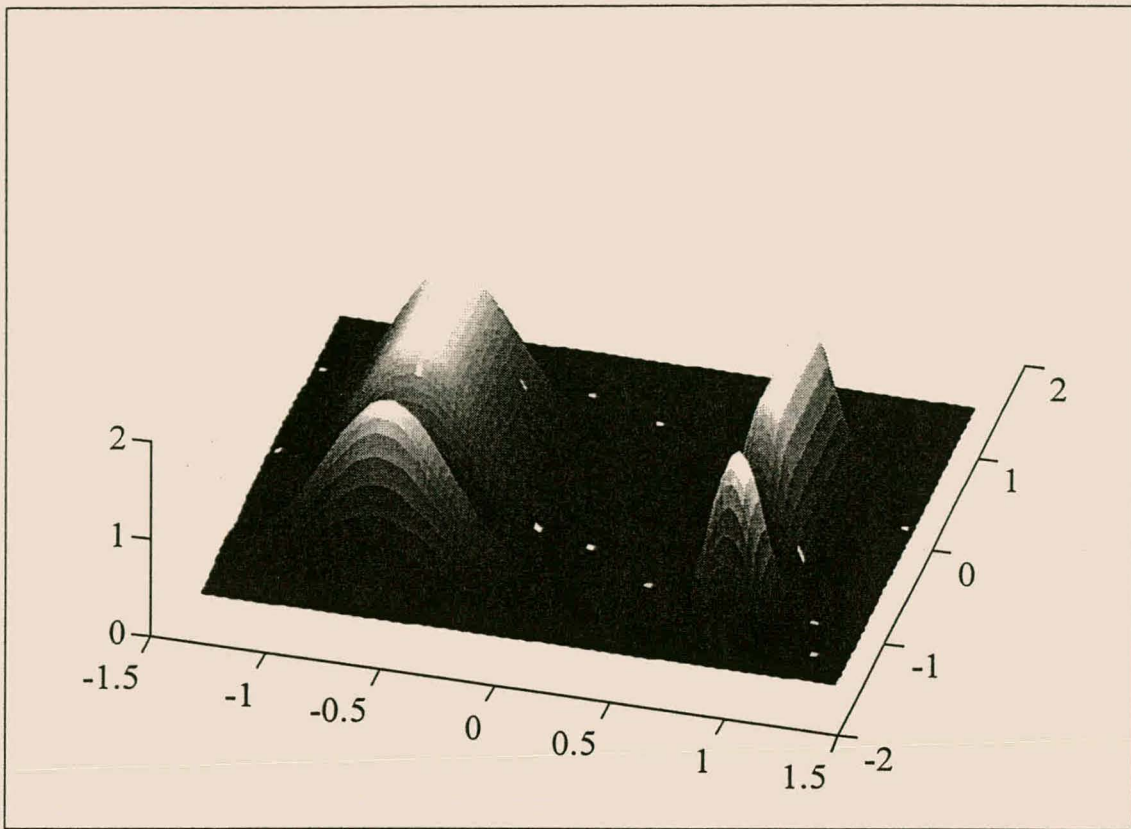


**Figure 6.1** Estimated Output of the GNG Fuzzy Rule Model



**Figure 6.2** Estimated Output of the CORA Fuzzy Rule Model





**Figure 6.3 Output Prediction Surface<sup>4</sup> of a Typical Fuzzy Rule Generated by the CORA Algorithm for the Spiral Problem.**

Second, the CORA model decision surface (figure 6.2) is generally more homogeneous and lighter than the GNG model decision surface (figure 6.1), especially in regions where the training data are sparse (i.e. towards the outskirts of the two spirals). This phenomenon can be attributed to the fact that consequent magnitude penalisation is performed. Such penalisation forces the CORA algorithm to build rule models with a relatively smooth output prediction surface. In contrast, no such penalisation is performed during GNG algorithm training. The GNG algorithm is therefore able to generate a spikier and less smooth decision surface.

### 6.1.2 Fuzzy Rule Model Predictive Accuracy

The next aspect of the CORA algorithm that will be discussed is the predictive accuracy of the CORA-generated rule model. As described in section 5.2.3, predictive accuracy is measured in terms of the root mean squared error for regression problems and the percentage of correctly classified data exemplars for classification problems. The problems that were considered for this

---

<sup>4</sup> The few randomly distributed white dots should again be ignored. They have nothing to do with the actual output prediction surface.



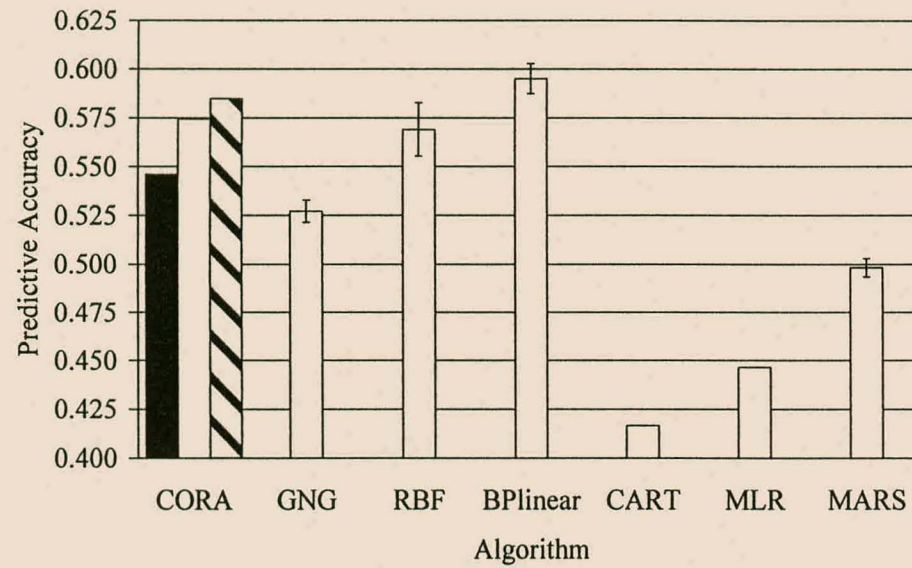
investigation were the Abalone, Auto, Housing, Servo and Sincos regression problems. In addition, the Ionosphere, Slugflow, Pima and WBC classification problems were also studied.

The results for this investigation are presented in figures 6.4 through 6.12. Only results on validation data are given. For algorithms such as CART or BEXA that perform tree or rule model postpruning, the validation results of the pruned models are presented. In each figure the first three bars or columns give worst, median and best predictive accuracy obtained by the CORA algorithm, averaged<sup>5</sup> over the entire set of experiments performed for this chapter. The results presented are those obtained by the CORA algorithm after all three algorithmic components (RTS, MRG and RED) have been applied. The remaining bars in each figure give the results<sup>6</sup> for the other algorithms that were studied.

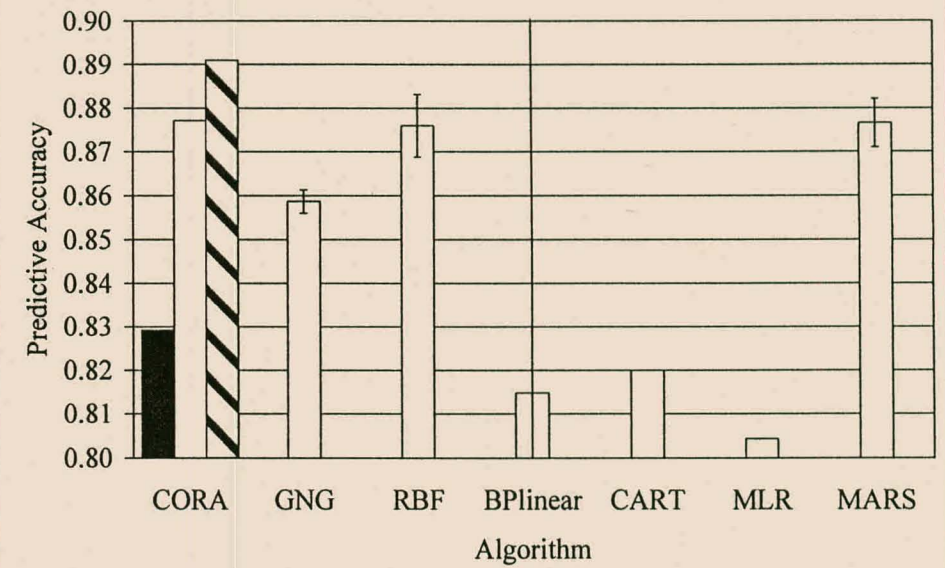
---

<sup>5</sup> In particular, for each set of training parameters investigated, the average over three different random seed runs was first calculated. Thereafter the average over all of these initial average results was calculated. These latter results are the ones presented in the figures.

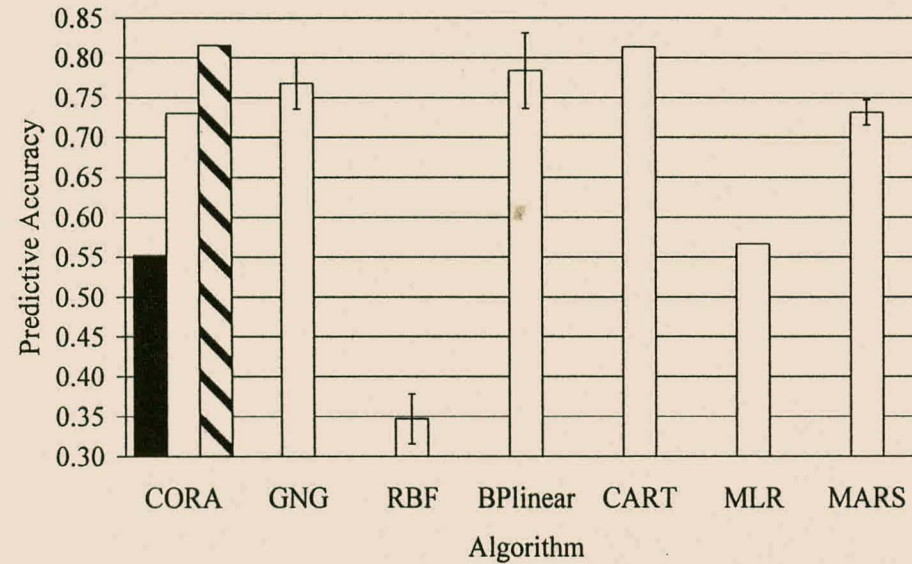
<sup>6</sup> For stochastic (i.e. that contain a probabilistic component) algorithms the magnitude of each bar is the average over each of the three different random seed runs performed. The error lines for these bars each represent one standard deviation of the applicable results above and below the average.



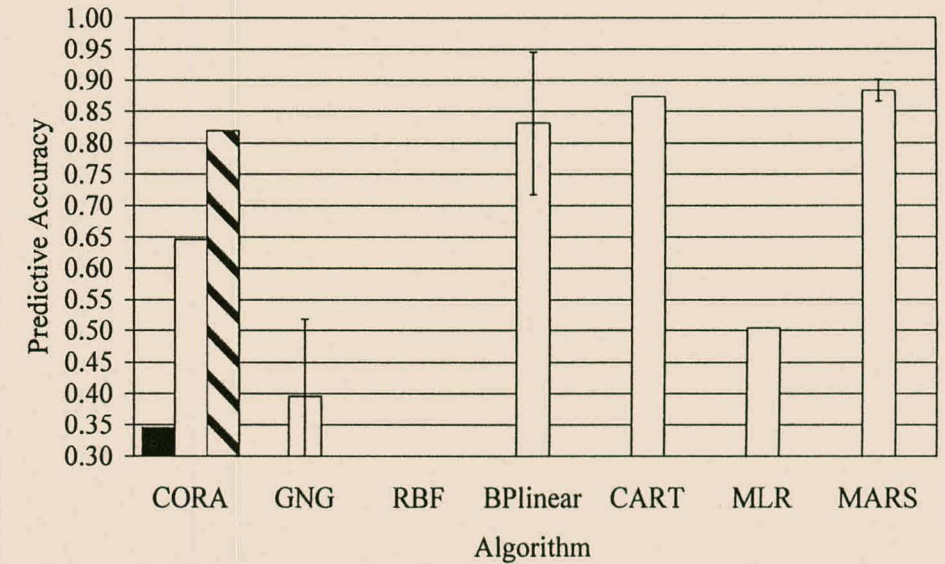
**Figure 6.4 Predictive Accuracy on the Abalone Problem**



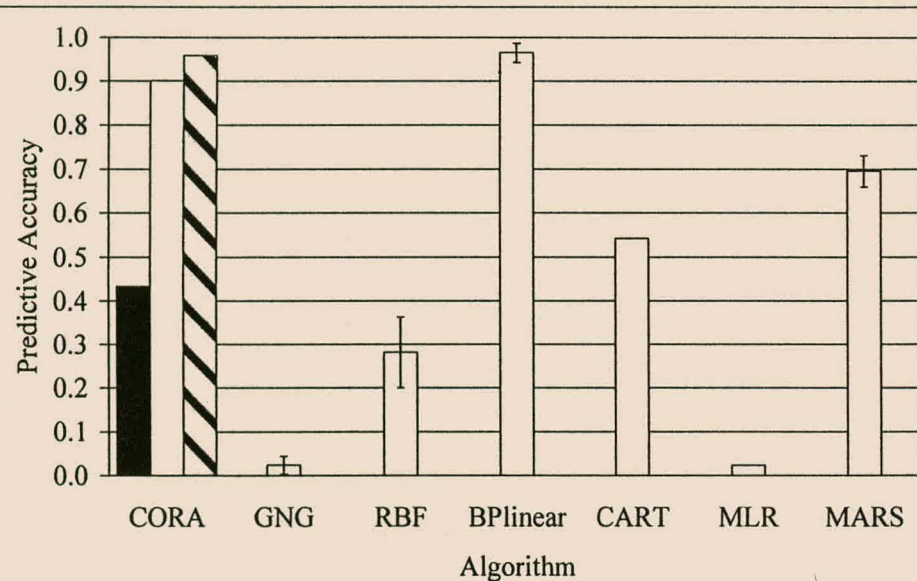
**Figure 6.5 Predictive Accuracy on the Auto Problem**



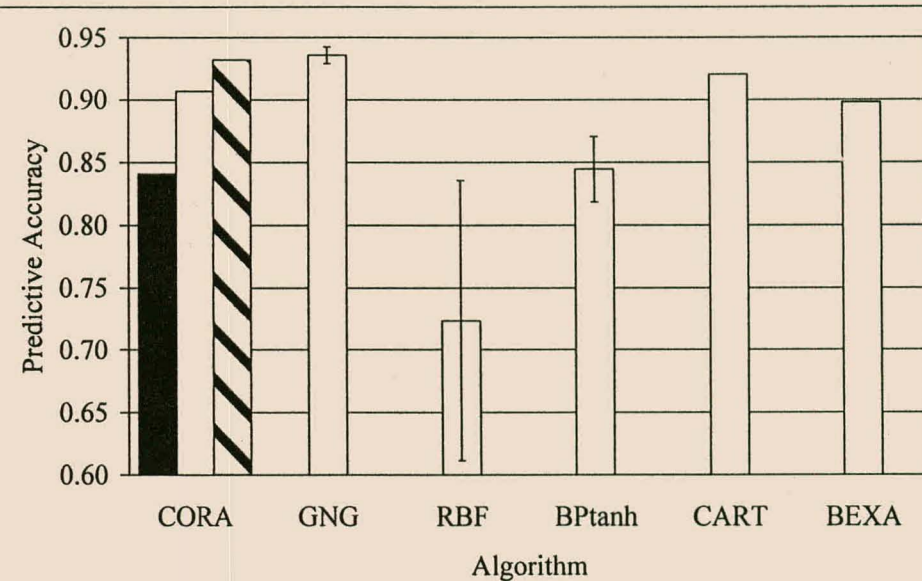
**Figure 6.6 Predictive Accuracy on the Housing Problem**



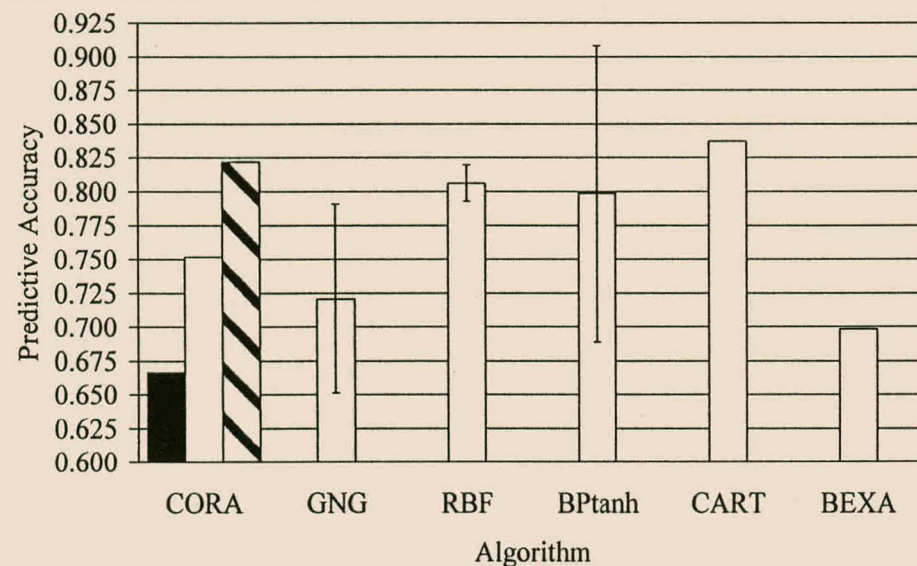
**Figure 6.7 Predictive Accuracy on the Servo Problem**



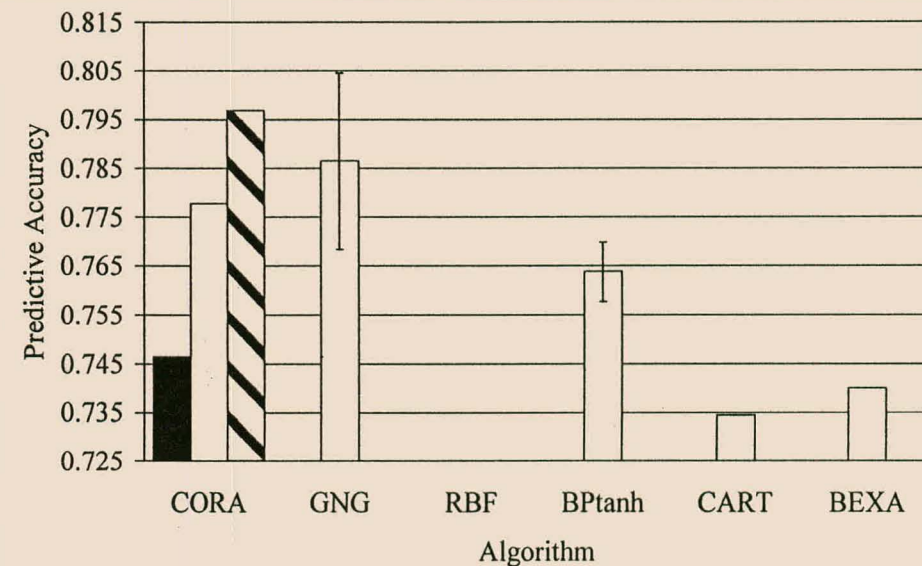
**Figure 6.8 Predictive Accuracy on the Sincos Problem**



**Figure 6.9 Predictive Accuracy on the Ionosphere Problem**

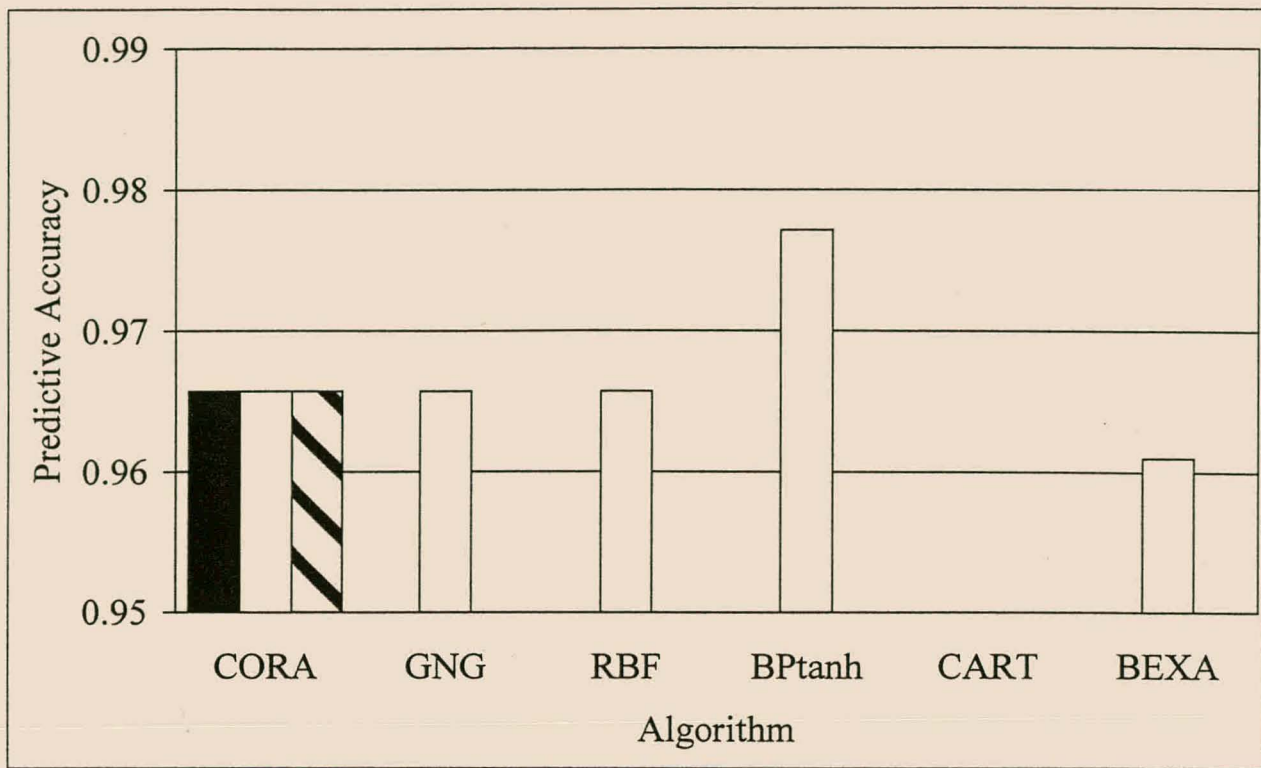


**Figure 6.10 Predictive Accuracy on the Slugflow Problem**



**Figure 6.11 Predictive Accuracy on the Pima Problem**





**Figure 6.12 Predictive Accuracy on the WBC Problem**

The labels for the radial basis function network, the multilayer perceptron with a summation output node, the multilayer perceptron with a tanh output node, multivariate linear regression and finally the MARS algorithm are respectively RBF, BPlinear, BPtanh, MLR and MARS. Note that meaningful results could not be obtained in the case of the RBF algorithm on the Servo and Pima problems as well as the CART algorithm on the WBC problem. Note also that the high standard deviation of the results obtained by the RBF, BPlinear and GNG algorithms can be attributed to the fact that for some random seeds these algorithms constructed significantly suboptimal models. This was as a result of the training techniques of these algorithms getting stuck in local optima in the solution space, which led to the large variation in model performance for different random seeds.

Inspection and analysis of the results in figures 6.4 through 6.12 reveal the following. First, in general the CORA algorithm was capable of constructing rule models with comparable or seemingly better<sup>7</sup> accuracy than all the other techniques considered, bar the two multilayer

<sup>7</sup> As described in section 5.2.2 the difference in results obtained by the models generated by the various algorithms are not compared on a strict statistical basis in this chapter. However, in order to be able to summarise the results presented in the figures, less stringent comparisons between the performance of the various models are made. These should be interpreted as such.

perceptron variants. This is especially true for the regression-type problems for which the CORA algorithm is designed. The principal exception is the Servo problem. For this particular problem the CORA-derived models are seemingly worse than those generated by the BPlinear, CART as well as MARS algorithms. In addition, the various CORA models display large variation in predictive performance.

The relatively poor performance by the CORA models can be attributed to two factors. First, the Servo problem contains 83% discrete attributes. As described in section 5.3.1.6 the discrete attributes of this problem were encoded using ten new attributes and binary encoding before presentation to any of the algorithms. It seems that the GNG algorithm (upon which the CORA algorithm is partly based) in particular, struggles to solve problems for which the attribute space is primarily discrete. As described in chapter 4, the GNG algorithm attempts to construct a topology preserving mapping of the manifold formed by the training data in attribute space. The lack of numeric attributes and the consequent polarisation of exemplars in the attribute space make it very difficult for the GNG algorithm to generate such a mapping.

The second reason why the CORA algorithm performs poorly on the Servo problem can be ascribed to the lack of training data and consequent sparsity of data in the high-dimensional attribute space. This latter phenomenon is commonly referred to as the curse of dimensionality (Bellman, 1961). The discrete attribute encoding used on the Servo problem in this study exacerbates this problem. It is therefore likely that overfitting occurred, i.e. the CORA algorithm constructed a model that modelled the training data rather than the underlying manifold that the training data are meant to be representative of. In addition, there are only 42 validation exemplars (table 5.1). Poor prediction of just a few validation exemplars will therefore have a pronounced effect on validation accuracy.

Next, consider the difference in results obtained by the CORA algorithm and the GNG algorithm. From figures 6.4 through 6.12 it can be seen in eight out of nine problems (the exception is the Ionosphere problem) the best CORA model performance exceeds or is the same as the average accuracy of the GNG algorithm upon which the model was based. In addition in six out of nine problems the median accuracy obtained by the various CORA models exceeds or is the same as the average accuracy of the GNG models. The exceptions are the Housing, Ionosphere and Pima problems, two of which are classification problems. It seems therefore that the CORA modelling language that encompasses internally disjunctive fuzzy rules is generally more powerful than the modelling language used by the GNG algorithm (which does not allow



internally disjunctive rules), even if a range of different CORA algorithm training parameters are used.

The above statement is aptly illustrated by the experiments performed on the Sincos problem. The best results that were obtained using the GNG algorithm are based on a hidden layer size of 120 cell nodes<sup>8</sup>. For such a GNG network the training  $R^2$  is 0.83 with a corresponding validation  $R^2$  of 0.52. In contrast, the CORA algorithm was able to construct fuzzy rule models with a median validation  $R^2$  of 0.90 (best validation  $R^2$  is 0.96) based on a GNG model containing only 30 nodes. (All CORA experiments were performed based upon a GNG model containing only 30 nodes.) However, the best model that the GNG algorithm could build if only 30 rules were used, exhibited a training  $R^2$  of 0.24 and a validation  $R^2$  of 0.024. Therefore the CORA algorithm was able to construct more accurate models in comparison to the GNG algorithm, even if the GNG algorithm was allowed to use four times as many cell nodes as was allowed the CORA algorithm.

More in depth inspection of the models derived by the GNG and CORA algorithms for the Sincos problem brought the following to light. The restrictions on the type of fuzzy rules that the GNG algorithm can construct (no internal disjunction allowed) makes it difficult for the algorithm to effectively model the primarily sinusoidal nature of the Sincos output. In contrast, the ability of the CORA algorithm to build internally disjunctive rules allows the algorithm to selectively use membership functions in the appropriate attribute dimensions to approximate the sinusoidal output. Membership functions from the same original GNG radial basis functions from which the above membership functions were obtained, but that represent different attributes, can be used to model the secondary linear nature of the Sincos output surface.

In addition, it was found that the CORA algorithm was forced to use relatively large fuzzy rule consequents in order to build accurate rules from the very inaccurate 30 fuzzy rules obtained from the GNG algorithm. In particular, rule consequent values ranged between +27.7 and -33.0. In normal circumstances, given the preprocessing performed on the data, such large and small consequents imply that the predicted output surface is fairly nonsmooth or irregular. Such variation is normally avoided whenever possible. Owing to this, as well as the unique means by which the CORA algorithm solved the Sincos problem, the Sincos problem will not be studied hereafter.

Next, a comparison of the results obtained by the CORA algorithm and the RBF technique shows that the use of additional mathematical constructs, such as those used by the RBF technique (see section 5.3.2.2 for details), does not necessarily result in the most accurate models. The results presented in figures 6.4 through 6.12 show that the CORA algorithm, which assembles 0<sup>th</sup>-order Sugeno rules with possible internal disjunction, can obtain comparable or seemingly better results than the RBF algorithm.

Finally, consider the results obtained for the WBC problem (figure 6.12). This problem has been studied in various forms by a number of researchers, e.g. Curram and Mingers (1994), Brodley and Utgoff (1995), Dougherty, et al. (1995), Kohavi and John (1997), Domingos (1996c), Domingos and Pazzani (1996), Theron (1994) and Hamker and Heinke (1997), with the purpose often being to compare new algorithms with existing ones. The WBC problem was investigated in this chapter for the same reason. However, initial results showed that if a 75% / 25% split of the randomised set of problem data was used (as was done for most other problems), then validation classification accuracy results very similar to the best published results could be obtained using very small models. In particular, the CORA, GNG and RBF algorithms required only two hidden layer nodes and practically no training parameter tuning in order to obtain a validation classification accuracy of 0.965. Owing to the very small models that were built it was decided not to further evaluate the CORA algorithm on this problem for sections of this chapter that examine the influence of CORA algorithm training parameters. In addition, the relative ease by which good validation results can be obtained makes future study of this problem, as a means comparing the predictive performance of two or more techniques with each other, questionable. In the author's opinion, this data set is only useful as a problem for algorithm development.

### 6.1.3 Fuzzy Rule Model Complexity

This section compares the complexity of the models derived by the CORA algorithm with those models generated by other algorithms. The problems that are studied in this section are the Abalone, Auto, Housing and Servo regression problems as well as the Ionosphere, Slugflow, Pima and WBC classification problems.

As described in section 5.1.1, model complexity is defined in terms of a number of factors. These are the number of rules that make up the model, the number of concepts used to form the

---

<sup>8</sup> The results for both the GNG algorithm and CORA algorithm models that are presented in figure 6.8 are based on a GNG network hidden layer size of 30 cell nodes.

rules, the number of parameters required to define the model and the average number of rules that need to be evaluated to determine the output of the model. As mentioned in section 6.1.1, the last factor can also be interpreted as the average overlap of the set of rules in attribute space. Not all algorithms considered in this comparison exclusively derive “if...then...” type rules and some don’t derive such rules at all. For this reason only those models are included in the evaluation that can be realistically compared to each other with respect to the particular complexity measure under consideration.

Specifically, for the “number of rules” and “number of concepts” measures only the models generated by the CORA, GNG, CART and BEXA algorithms are compared because only these algorithms generate models consisting exclusively of rules with each rule containing no mathematical constructs. For the “number of parameters” complexity measure the CORA, GNG, RBF, BPlinear, BPtanh, MLR and MARS models are compared with each other, where applicable. Finally, for the “attribute space overlap” measure the CORA, GNG, CART and BEXA models are compared with each other.

It must be noted at this stage that the definition of the “number of parameters” complexity measure is biased against the radial basis function neural network-based algorithms. The definition of this measure (see section 5.1.1) was originally created with the aim of comparing the relevant complexities of the CORA and GNG rule models. The ways in which other algorithms calculate this measure was not considered at that stage. Post-experimental analysis of results brought to light that some algorithms assume that if exactly the same parameter is used in two or more different contexts or places in a model, the two or more parameters are assumed to be one parameter only in the calculation of model complexity.

For example, the definition given in section 5.1.1 assumes that each membership function that is extracted from a  $n$ -dimensional, hyperspherical Gaussian is uniquely described by two parameters, viz. a centre and a width. According to the definition of the “number of parameters” complexity measure,  $n$  membership functions therefore require  $2n$  parameters to be defined. However, by definition the width of a hyperspherical Gaussian is the same in all dimensions. In other words, strictly speaking,  $n$  membership functions extracted from a  $n$ -dimensional hyperspherical Gaussian require the specification of  $n + 1$  parameters only, viz.  $n$  centres and a single width.

In addition, it must also be noted that this complexity measure does not take into account the functions for which the parameters are used. For example, no distinction is made between parameters used for one-dimensional Gaussian functions and those used for regression splines.

Different forms of parameterised functions have inherently different complexities. The same argument holds for the “number of concepts” complexity measure. A concept such as a membership function is, in the author’s opinion, more complex than a crisp singleton used in a crisp rule.

Owing to the difficulty of calculating the number of model parameters (for CORA models in particular) based upon the above-described assumption (that different parameters in different places can in fact be identical and should be counted as a single parameter) and of quantifying the complexity of parameterised functions, the complexity measure defined in section 6.1.1 is still used in this chapter.

The comparative results related to model complexity based on the “number of rules” measure are summarised in figures 6.13 through 6.16. Those results based on the “number of concepts” measure are presented in figures 6.17 through 6.20 and those based on the “number of parameters” measure are given in figures 6.21 through 6.24. Finally, those results based on the “attribute space overlap” complexity measure are presented in figures 6.25 through 6.28.

In each of these figures the best, median and worst results<sup>9</sup> are given for the CORA algorithm. The results presented in these figures were determined using all three CORA algorithm components (the RTS, MRG and RED components). The best, median and worst results were determined in an identical fashion to the way in which the results related to the predictive accuracy of the CORA model were calculated (described in section 6.1.2). Note that in this section “best” implies smallest, or least. In other words a model consisting of fewer “if...then...” rules is considered less complex, and therefore better, than a model consisting of more rules.

---

<sup>9</sup> Each figure contains the results for two problems. For instance, figure 6.13 presents the results for the Abalone problem as well as the Auto problem. The symbols in parentheses after each algorithm label on the horizontal axis of each graph distinguish between the results for the two problems. Ab stands for the Abalone problem, Au for Auto, H for Housing, S for Servo, I for Ionosphere, S for Slugflow, P for Pima and finally W for the WBC problem.

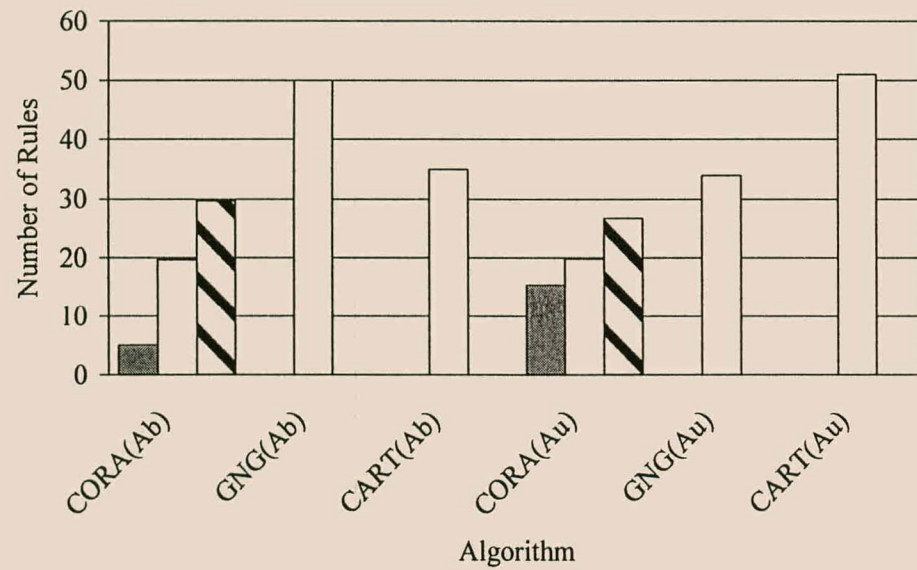


Figure 6.13 Number of Rules on Abalone and Auto Problems

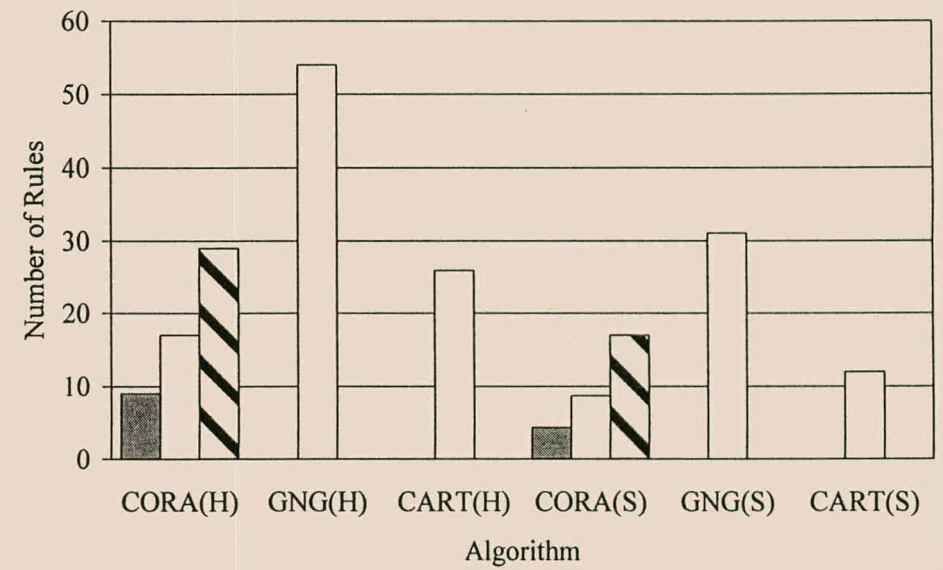


Figure 6.14 Number of Rules on Housing and Servo Problems

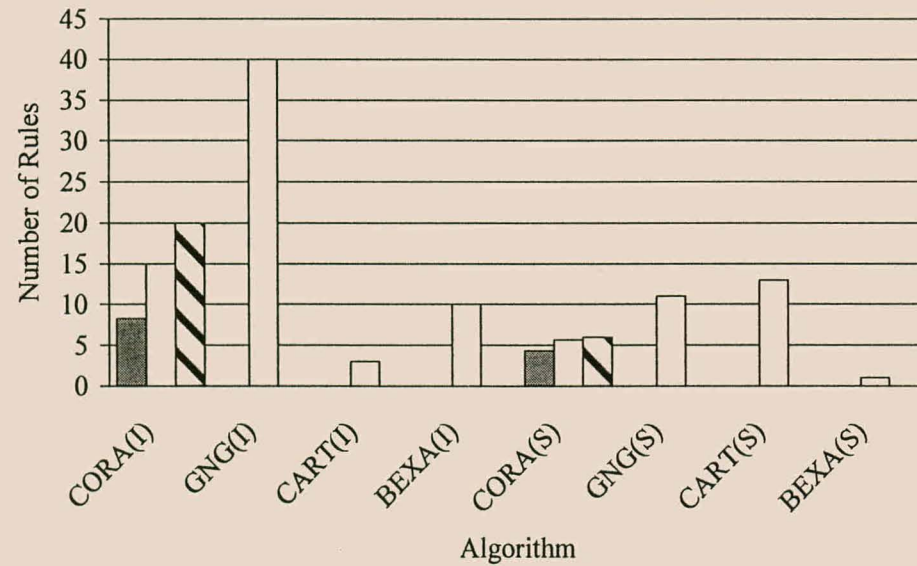


Figure 6.15 Number of Rules on Ionosphere and Slugflow Data

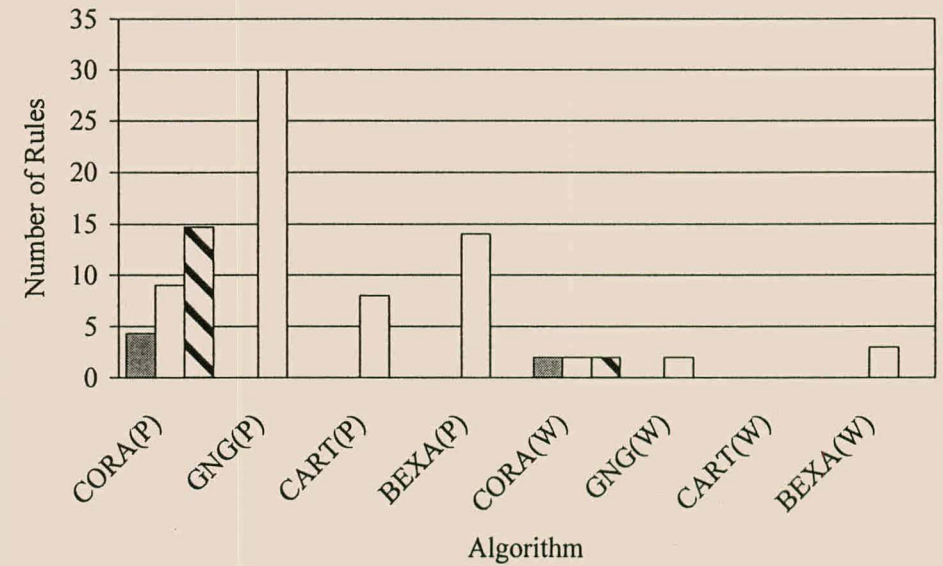
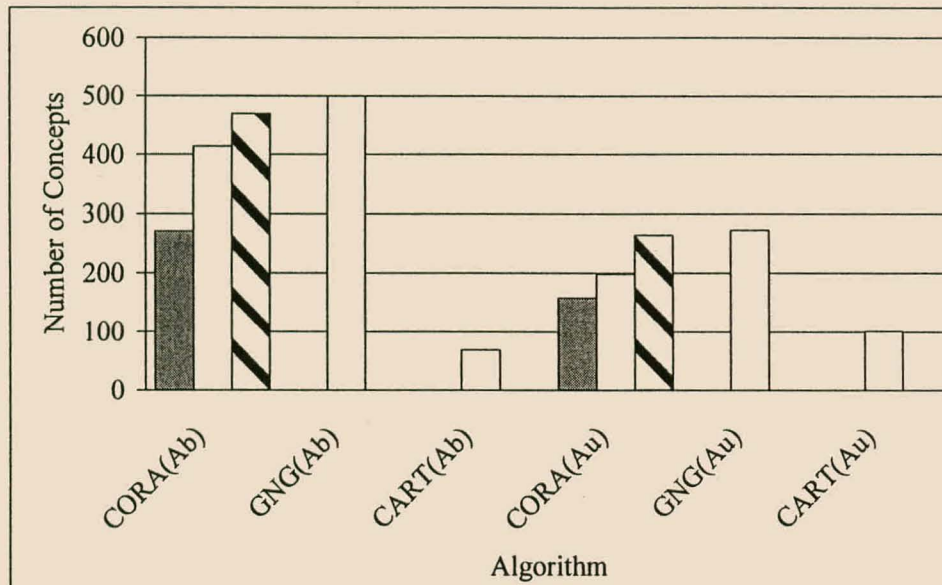
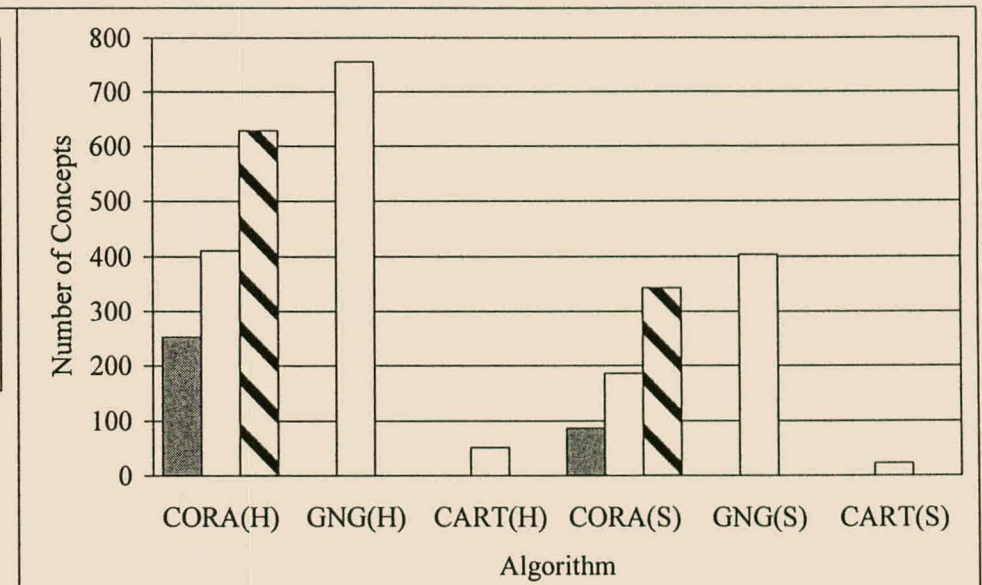


Figure 6.16 Number of Rules on Pima and WBC Problems

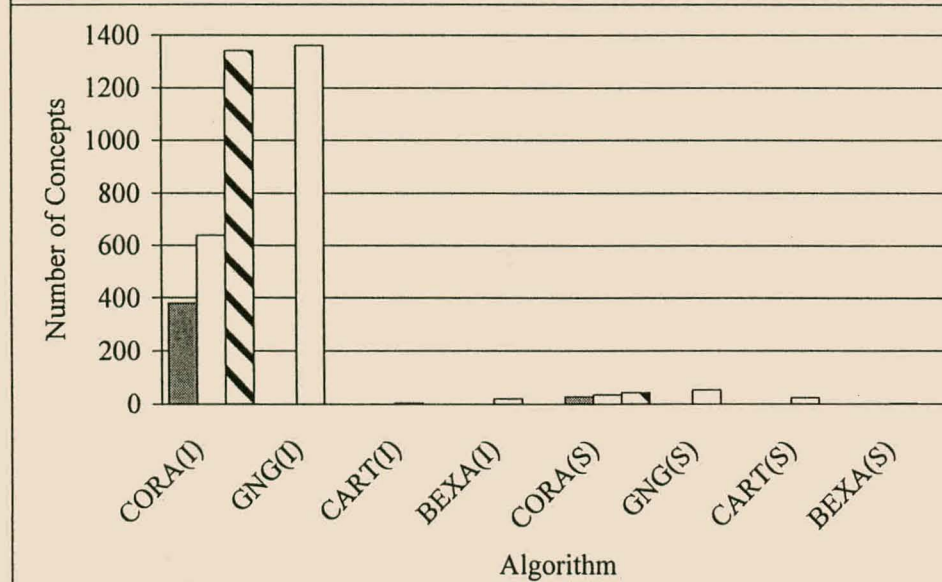




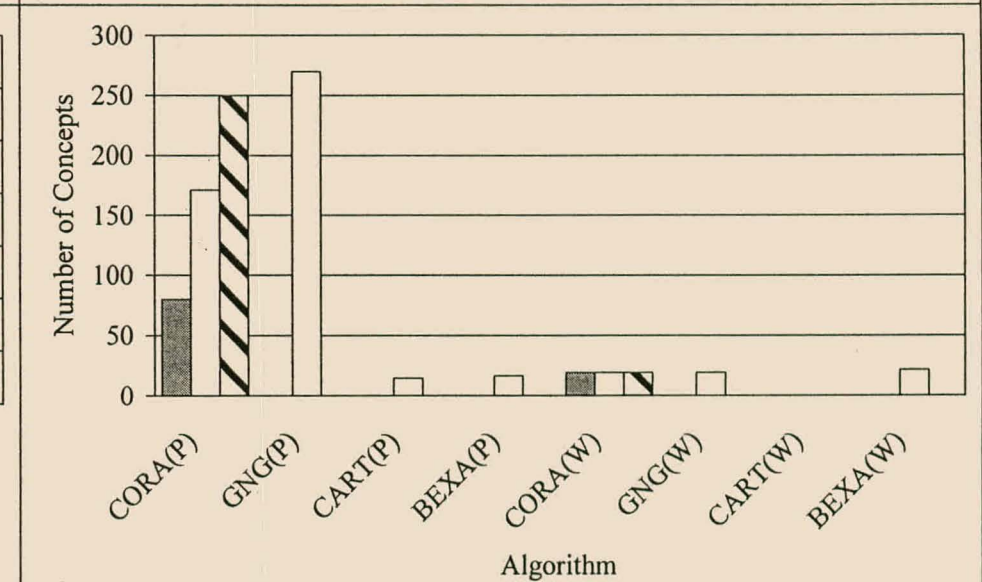
**Figure 6.17** Number of Concepts on Abalone and Auto Problems



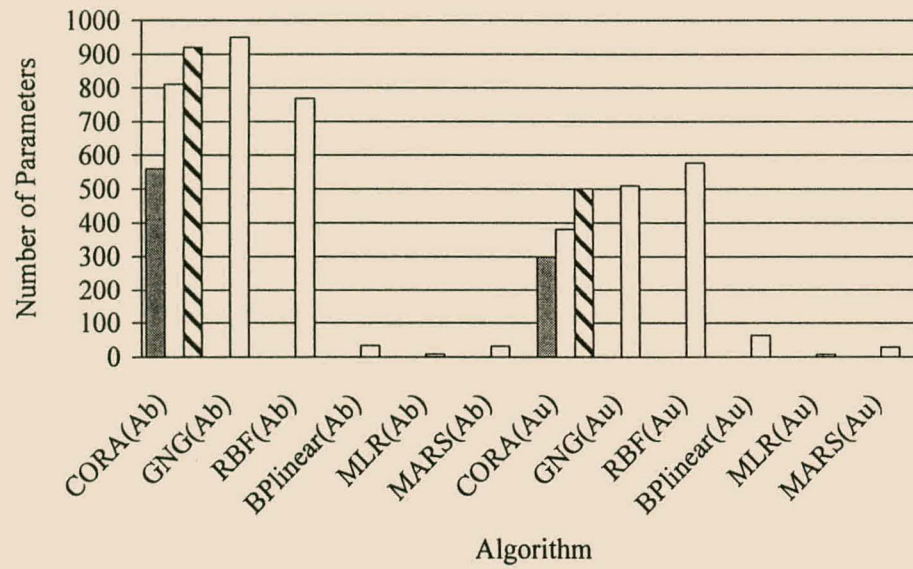
**Figure 6.18** Number of Concepts on Housing and Servo Problems



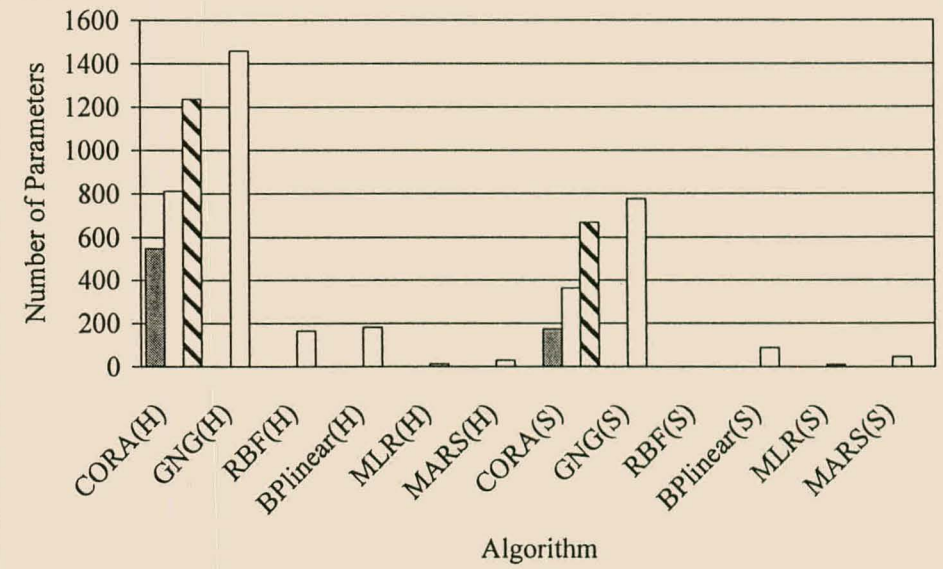
**Figure 6.19** Number of Concepts on Ionosphere and Slugflow Data



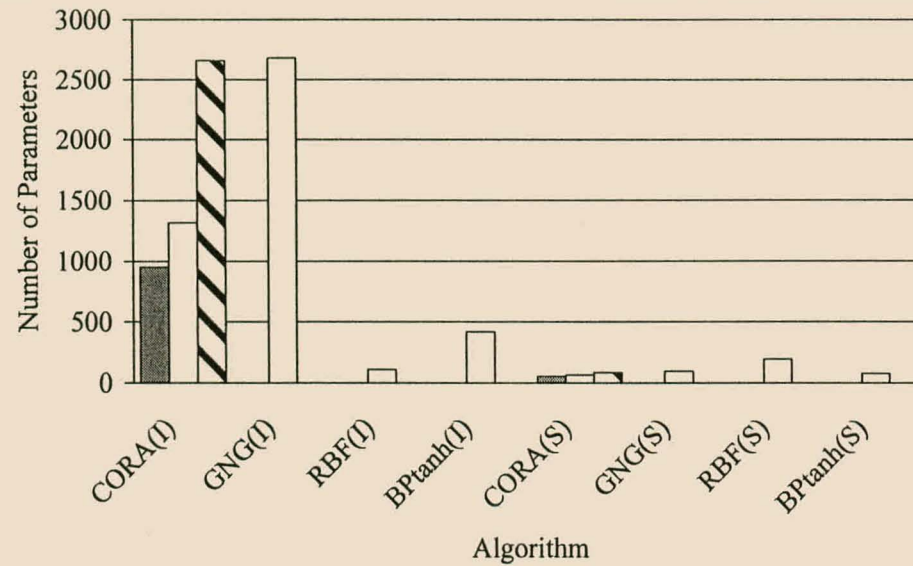
**Figure 6.20** Number of Concepts on Pima and WBC Problems



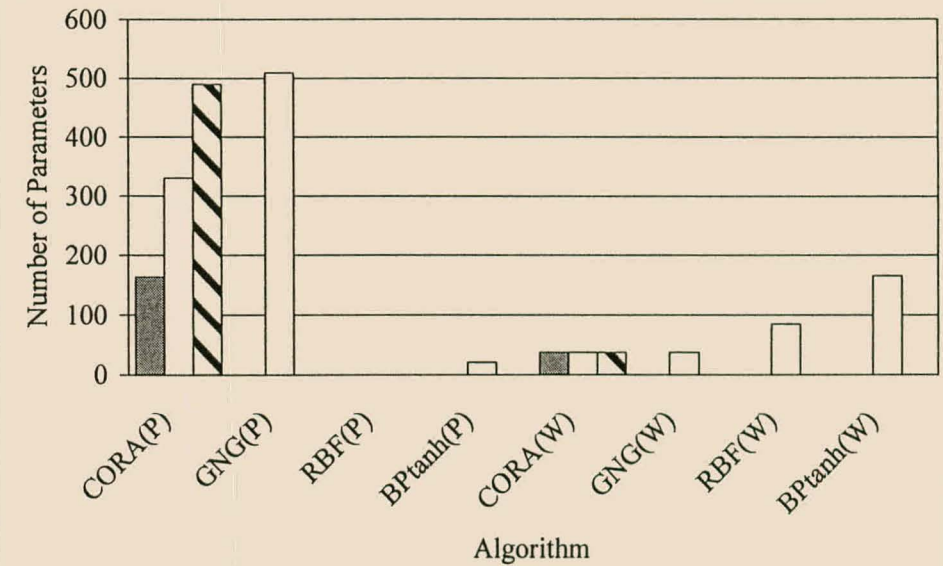
**Figure 6.21 Number of Parameters on Abalone and Auto Problems**



**Figure 6.22 Number of Parameters on Housing and Servo Data**

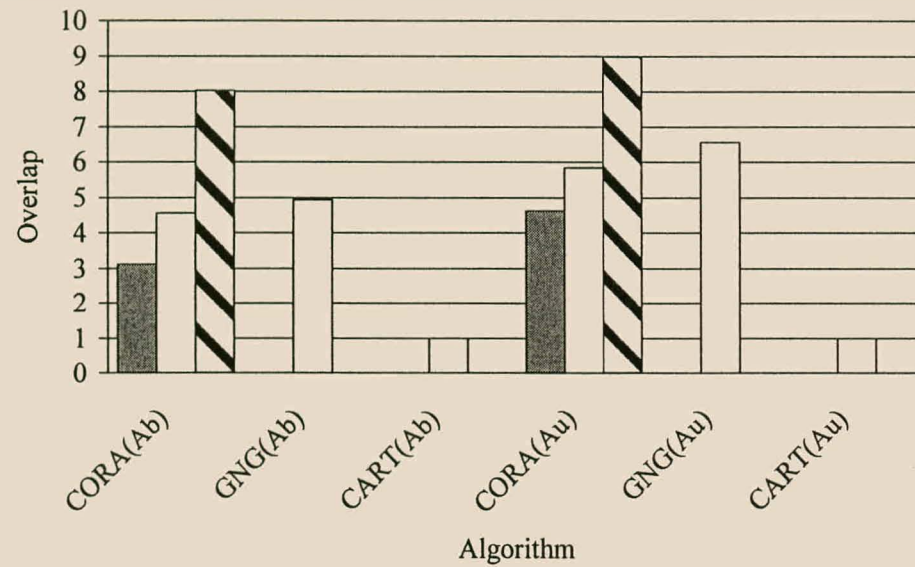


**Figure 6.23 No. of Parameters on Ionosphere and Slugflow Data**

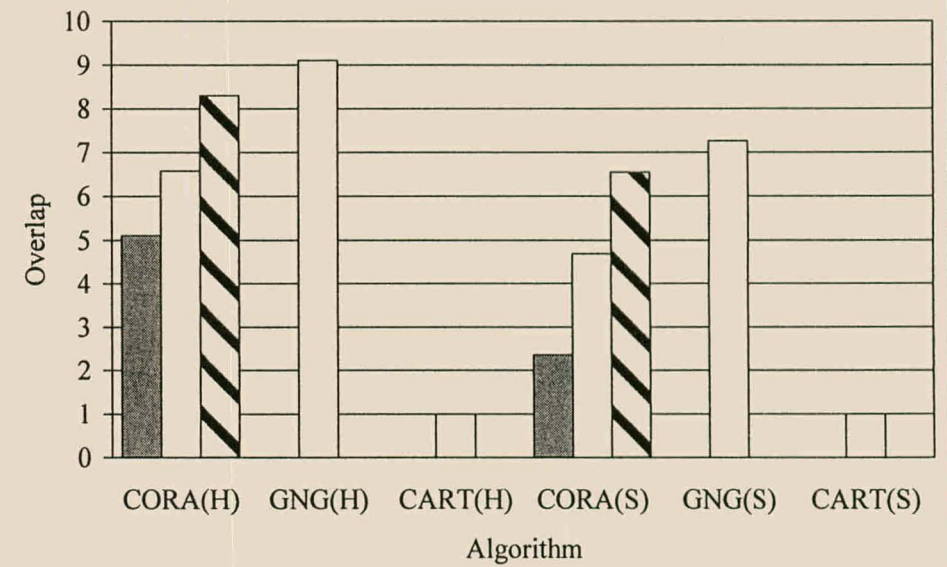


**Figure 6.24 Number of Parameters on Pima and WBC Problems**

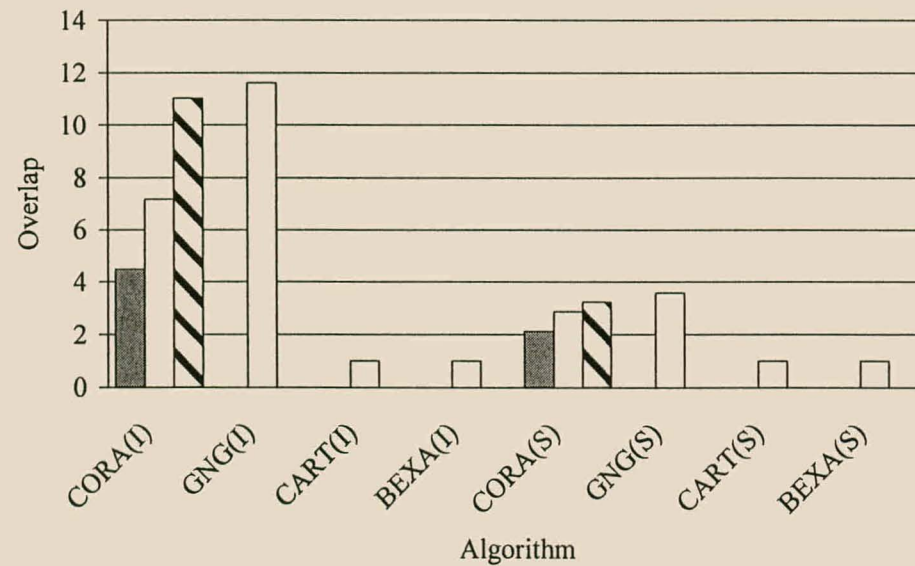




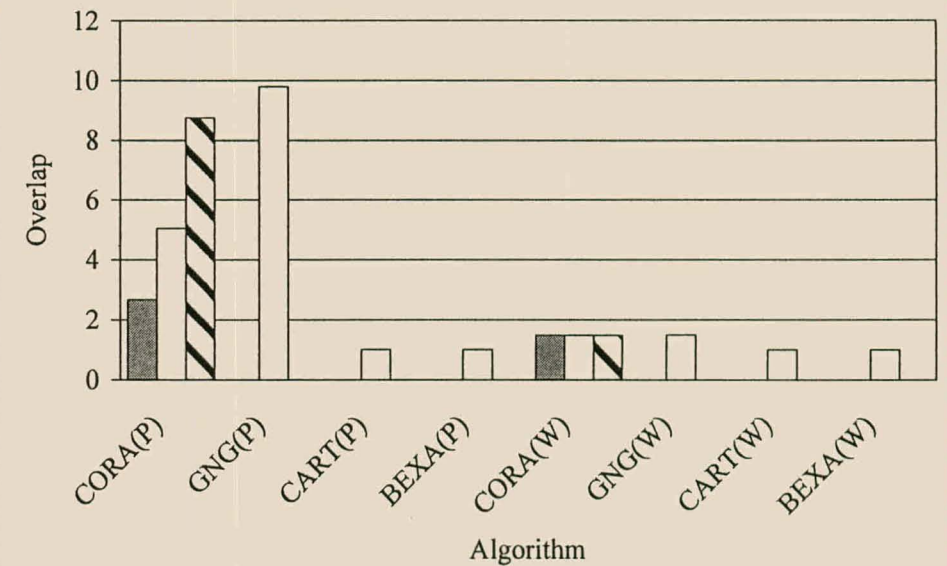
**Figure 6.25 Input Space Overlap on Abalone and Auto Problems**



**Figure 6.26 Input Space Overlap on Housing and Servo Problems**



**Figure 6.27 Input Space Overlap on Ionosphere and Slugflow Data**



**Figure 6.28 Input Space Overlap on Pima and WBC Problems**

The complexity results for the models generated by the other nondeterministic algorithms, viz. GNG, RBF, BPlinear, BPtanh and MARS, are the average results over the three different random seed runs performed using each algorithm. The complexity results for the CART and BEXA algorithms are based on the induced rule models after tree or rule pruning has been employed.

Consider first results for the “number of rules” complexity measure. For each of the four regression problems the best CORA models consist of fewer rules than both the GNG and CART models. In addition, inspection of figure 6.13 in conjunction with figures 6.4 and 6.5 shows that for the Abalone and Auto problems that no matter how many rules are used, the CORA models always seem to be more accurate than the CART models. This is also the case for the Pima classification problem (see figure 6.11 in conjunction with figure 6.16). However, for the remaining two regression problems (Housing and Servo), as well as the remaining classification problems (Ionosphere, Slugflow and WBC) the CORA models with the least rules are not guaranteed to be more accurate than their CART-derived counterparts. Finally, for the Ionosphere problem CART generated a model that is only seemingly slightly less accurate than the best (most accurate) CORA model but that has twice as few rules as the smallest (least rules) CORA model.

In comparison to the GNG-generated models, the CORA-derived models always used fewer rules, but this is by design. For the Abalone and Sincos problems (see figures 6.4 and 6.8 in conjunction with figure 6.13) this restriction on the number of CORA rules that could be assembled did not prevent the CORA algorithm from constructing more accurate models, however many rules could be assembled<sup>10</sup>.

For three of the four classification problems, viz. Ionosphere, Pima and WBC, CORA was able to construct models with fewer rules than those derived by BEXA. In terms of the Pima and WBC problems, the CORA algorithm was able to construct models with comparable or seemingly better predictive performance than those models induced by BEXA, no matter how many rules were used. In contrast, for the Slugflow problem the CORA algorithm could not construct as small (few rules) models as BEXA, but it was possible for the CORA algorithm to construct ostensibly more accurate models.

---

<sup>10</sup> For the Sincos problem the CORA algorithm was allowed to assemble five, ten or fifteen rules using the membership functions obtained from a 30-rule GNG model.

The “number of concepts” complexity measure is considered next. The results for this measure are summarised in figures 6.17 through 6.20. For all problems the CORA algorithm constructed models containing fewer concepts than the GNG-derived models, with the exception of the WBC problem where both types of models contained roughly the same number of concepts (owing to the simplistic nature of this problem). Again, this phenomenon is partly by design in that the CORA algorithm always uses at most the same number of membership functions as the GNG algorithm, but always constructs fewer fuzzy rules.

In terms of “number of concepts”, the CORA models (and therefore the GNG models) are, for all problems considered here, more complex than the models generated by both CART and BEXA. In the case of the Abalone, Housing, Servo, Ionosphere and Pima problems the CORA models are all at least twice as complex as those induced by CART or BEXA. The reason for the more complex CORA models is twofold. First, the CORA algorithm uses either single-sided or two-sided one-dimensional Gaussians to construct fuzzy rule antecedent parts. A typical antecedent is “the reactor temperature is *Low*”. In contrast, for linear attributes both CART and BEXA are able to use antecedents of the form “ $B_1 \leq b_1$ ” or “ $b_1 \leq B_1 \leq b_2$ ” where  $b_1$  and  $b_2$  represent linear attribute values of attribute  $B_1$ . Such latter types of antecedents are able to cover, i.e. hold true, for a much larger region in attribute space than a membership function described by a one-dimensional Gaussian. It is therefore possible for the CART and BEXA algorithms to use such antecedents to represent large regions of a linear attribute which in turn translates to simpler “if...then...” rules.

Second, the current CORA algorithm is forced to use all the membership functions generated by the GNG algorithm. It could be argued that this is not strictly necessary in order to guarantee accurate rule models, but this is what is currently done. The use of all available membership functions, coupled with the lack of explicit “number of concepts” complexity penalisation during the RTS phase of rule assembly, means that the RTS component will tend to generate accurate, but relatively complex models. It was planned that the MRG and RED components of the CORA algorithm would substantially reduce the complexity of CORA models to bring it more in line with that of the other types of rule models. Unfortunately this does not seem to have happened to the extent that was hoped for. This phenomenon will be studied in more detail when the influence of CORA training parameters is discussed later on in this chapter in section 6.3.2.



Figures 6.21 through 6.24 present the results<sup>11</sup> for the “number of parameters” complexity measure experiments. As with the “number of concepts” complexity measure, and for the same reasons, the CORA models always use less parameters than the GNG-derived models do.

The CORA models are generally appreciably more complex in comparison to the models generated by the other techniques (not the GNG algorithm) considered in this chapter. The exceptions are the following. For the Abalone and Auto problems (see figure 6.21) the CORA algorithm is able to build less complex models than the RBF algorithm. For the latter problem the CORA models always use less parameters than the corresponding RBF models although the predictive accuracy of the former is not always better than that of the latter (see figure 6.5 in conjunction with figure 6.21). Furthermore, for the Slugflow problem (see figure 6.23) the complexities of the CORA models (and the GNG models) are all less than that the RBF-generated models and roughly the same as that of the BPtanh model. For the WBC problem (figure 6.24) both the CORA and GNG algorithms generate models that are less complex than those generated by either the RBF or the BPtanh algorithm. These CORA and GNG models are as accurate as the RBF models but less accurate than the BPtanh models (see figure 6.12).

The final complexity issue that will be discussed is that of the average number of rules that need to be evaluated to determine model output, or in other words attribute space overlap. The results for these experiments are summarised in figures 6.25 through 6.28. Only the CORA, GNG, CART and BEXA algorithms are investigated, because only they exclusively construct “if...then...” rules.

As mentioned in section 6.1.1, both CART and BEXA generate models where the average number of rules that need to be evaluated to determine rule model output is always exactly one. In contrast it was found for all but one problem, the WBC problem, that more than one rule needs to be evaluated to determine overall GNG or CORA model output. For the WBC problem, both the GNG or CORA algorithms constructed two rules only. Owing to the fuzzy nature of these rules, only slightly more than one rule (on average) would need to be evaluated to determine final model output.

This higher (greater than one) input space overlap phenomenon of the GNG and CORA algorithms can be attributed to the following algorithmic properties. First, the GNG algorithm

---

<sup>11</sup> The software implementation of the MARS algorithm reports model complexity in terms of, amongst others, a so-called “effective number of parameters” complexity measure. These are the results reported here.

has no explicit means of quantifying and limiting input space overlap. In addition, as described in section 4.1.2, the width of a cell node Gaussian is calculated as the fraction  $\tau^{12}$  of the mean length of all connections emanating from the particular cell node. (See the *insert\_cell\_node* function described in detail in figure 4.8 for more information.) Such width determination can lead to wide Gaussians that overlap significantly with, or even completely cover, small (less wide) Gaussians that are near to the wide Gaussian. Second, the CORA algorithm is forced to use all GNG-generated membership functions, but to assemble fewer fuzzy rules than the GNG algorithm was allowed to use. Intuitively it should be clear that such apportionment of membership functions to fewer rules should typically lead to higher attribute space overlap.

As described in section 4.3.4.7, input space overlap is explicitly penalised by the CORA algorithm. Figures 6.25 through 6.28 in conjunction with figures 6.4 through 6.12 show that it is possible for the CORA algorithm to generate models with less attribute space overlap, but with better predictive performance than the GNG algorithm. In particular, for the Housing, Servo, Ionosphere, Slugflow and Pima problems the attribute space overlap of each CORA model generated for this chapter is always less than that of the corresponding GNG model. However, these CORA models are not always more accurate than the corresponding GNG models. In other words, attribute space overlap penalisation is able to decrease model complexity, but for some problems this can be to the detriment of model predictive performance.

## 6.2 Effect of CORA Algorithm Training Parameters

This section examines the effect of CORA algorithm training parameters on the quality of model that can be assembled by the CORA algorithm. The parameters that are investigated are the number of assembled fuzzy rules, the percentage of either swaps or moves that are evaluated during a given RTS training iteration, whether only swaps, moves or both swaps and moves are utilised during RTS training, the level of consequent magnitude penalisation, the level of attribute space overlap penalisation and finally whether either Yu or Zadeh fuzzy conjunction and disjunction operators are utilised.

The problems considered in this section are the Abalone, Auto, Housing, Servo, Ionosphere, Slugflow and Pima problems. Owing to space constraints, not all results are presented here. The

---

<sup>12</sup>  $\tau$  was set to one in all the experiments described in this chapter.

remaining results are presented in the appendices. The discussion of the results includes both sets of information.

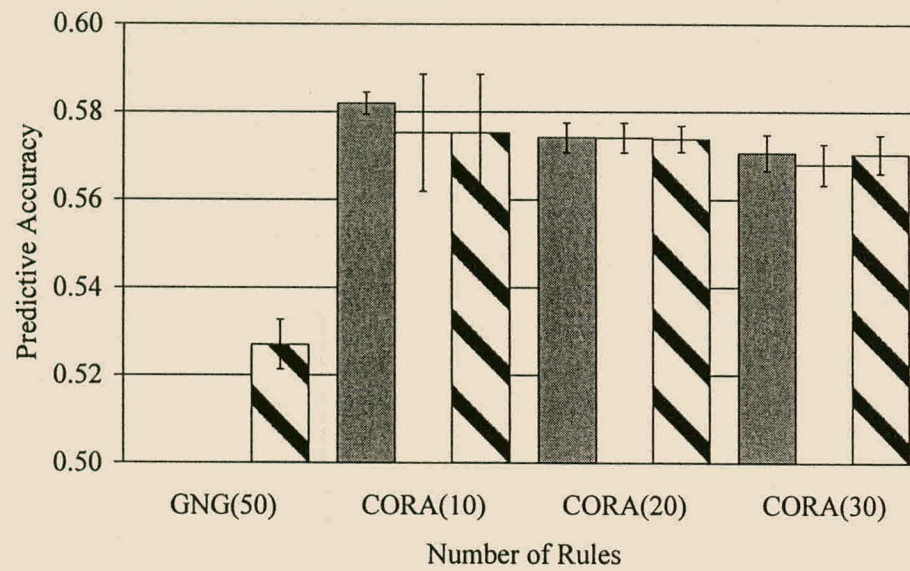
### 6.2.1 Number of Assembled Fuzzy Rules

Figures 6.29 through 6.40 show how the predictive accuracy and number of concepts of a set of CORA-derived fuzzy rules changes as the number of fuzzy rules that may be assembled by the algorithm changes. The results obtained by the GNG algorithm are also included. Appendix B presents similar results (the influence of number of CORA-assembled rules), but these are in terms of the number of rule model parameters.

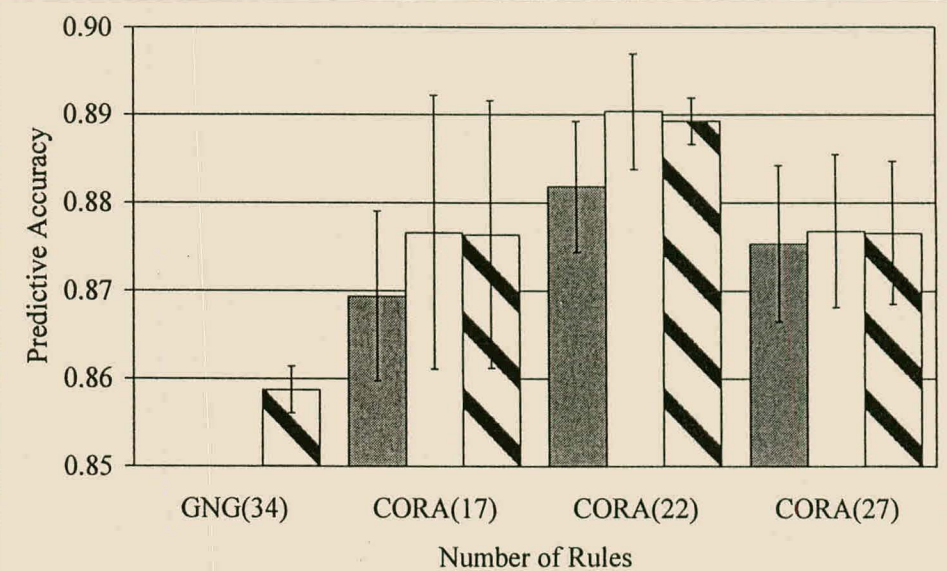
Each figure consists of ten bars. Each bar represents the average result over the three different random seed runs performed for the particular algorithm or algorithmic and for the particular training parameter settings. The black error line overlaid over each bar represents one standard deviation above and below the average result.

In each figure the first bar represents the results obtained by the GNG algorithm, as indicated by the horizontal axis label. The number in parentheses following this label (and all others on the horizontal axis) is the number of fuzzy rules used by the algorithm to generate the figure results. The following nine bars are all results obtained with the CORA algorithm. These nine bars are grouped in three groups of three bars each. For a particular group the first bar represents the results obtained directly after the RTS component of the CORA algorithm has completed execution. The second bar represents the results obtained after both the RTS and MRG components have been executed. Finally, the third bar represents the results after all three CORA components, viz. the RTS, MRG and RED components, have been employed.

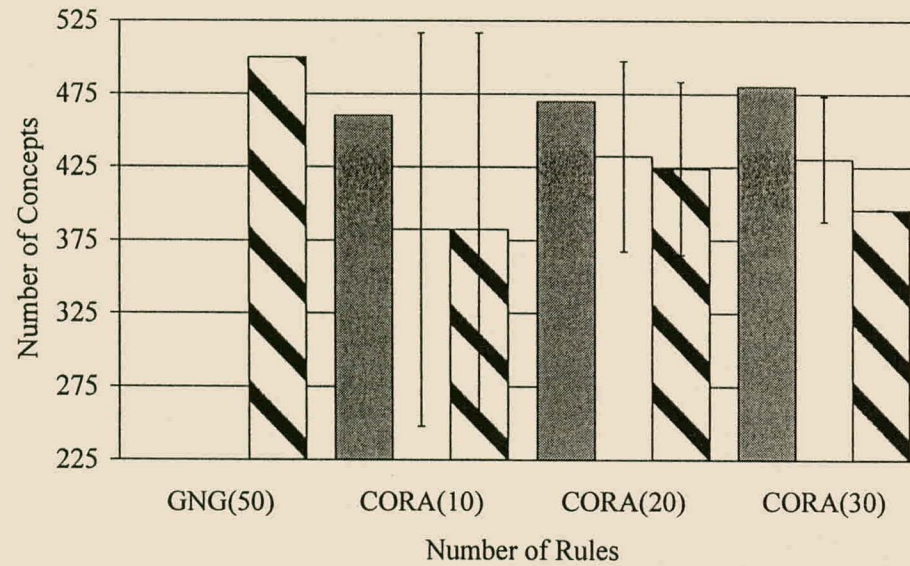
Finally, figures B.7 through B.24 in Appendix B present results of experiments that also studied changes in CORA model predictive accuracy and complexity as a function of the number of CORA-assembled rules. The difference between these results and those presented in figures 6.29 through 6.40 and figures B.1 through B.6 in Appendix B is that values of 0.5 and 0.01 for the `minimum_swap_move_threshold` training parameter were evaluated. The trends found in these latter results are in the author's opinion very similar to those found in figures 6.29 through 6.40 and Appendix B. For this reason they are not explicitly mentioned in the following discussion but for completeness sake are presented in the appendices.



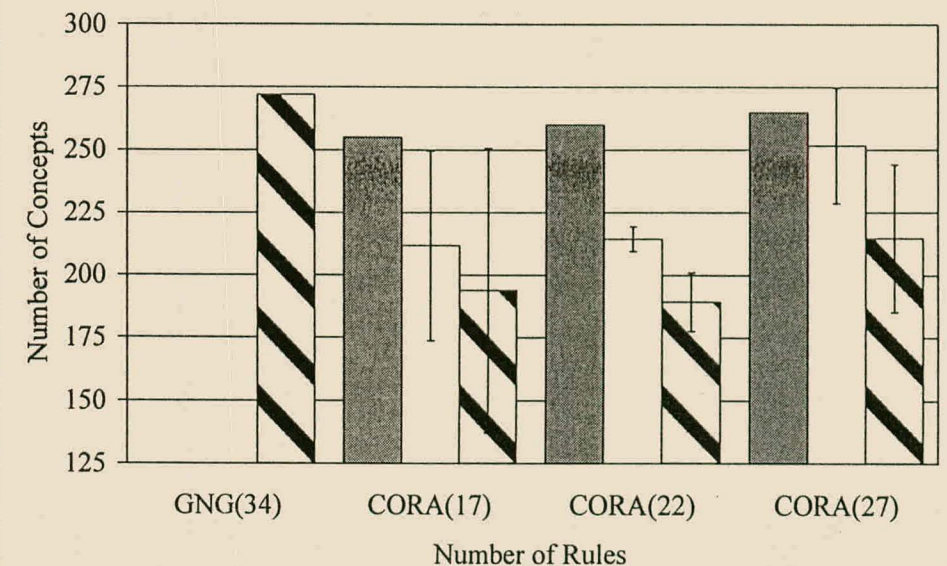
**Figure 6.29 Accuracy vs. Number of Rules for Abalone Problem**



**Figure 6.30 Accuracy vs. Number of Rules for Auto Problem**

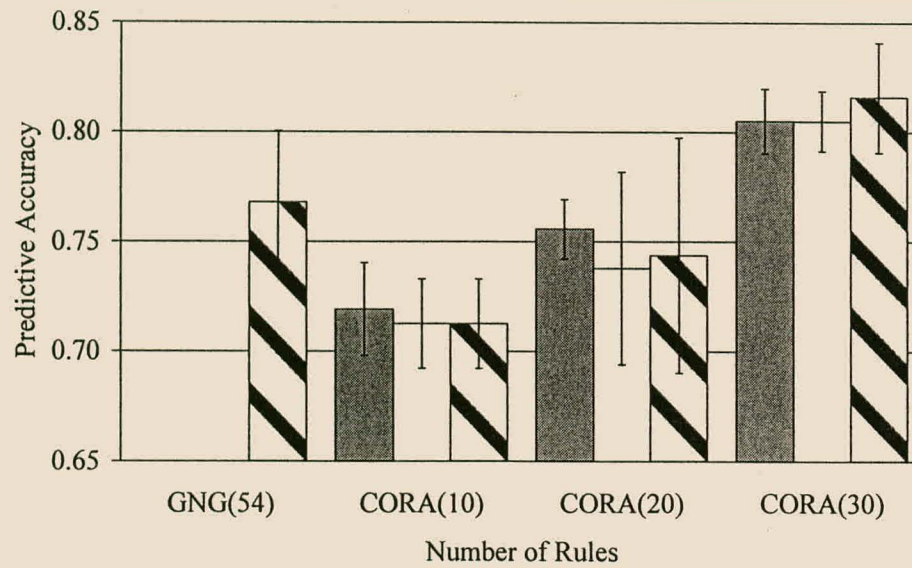


**Figure 6.31 No. of Concepts vs. No. of Rules for Abalone Problem**

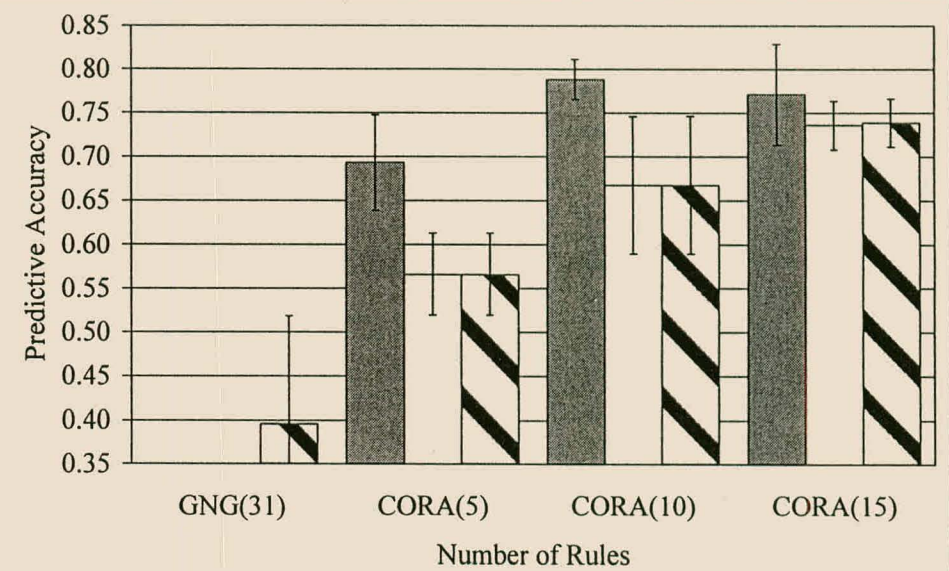


**Figure 6.32 No. of Concepts vs. No. of Rules for Auto Problem**

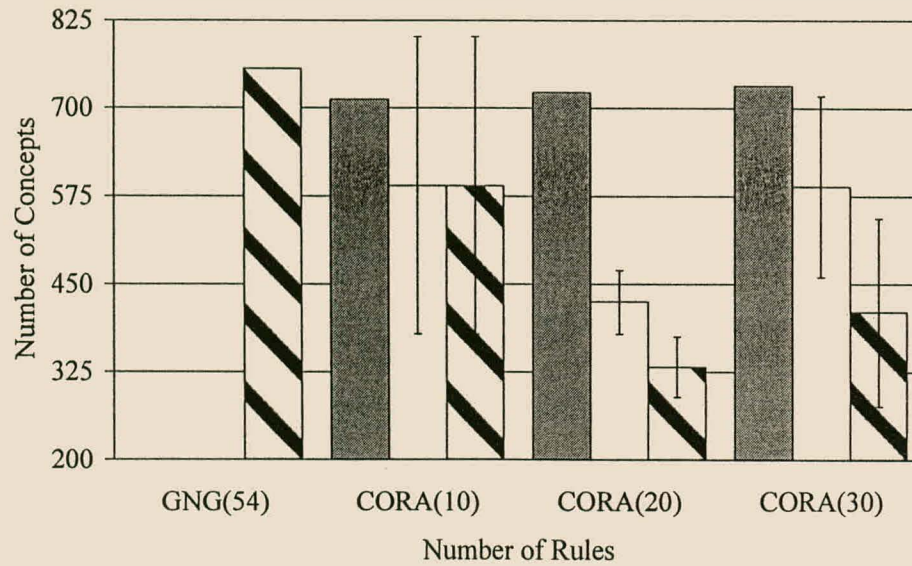




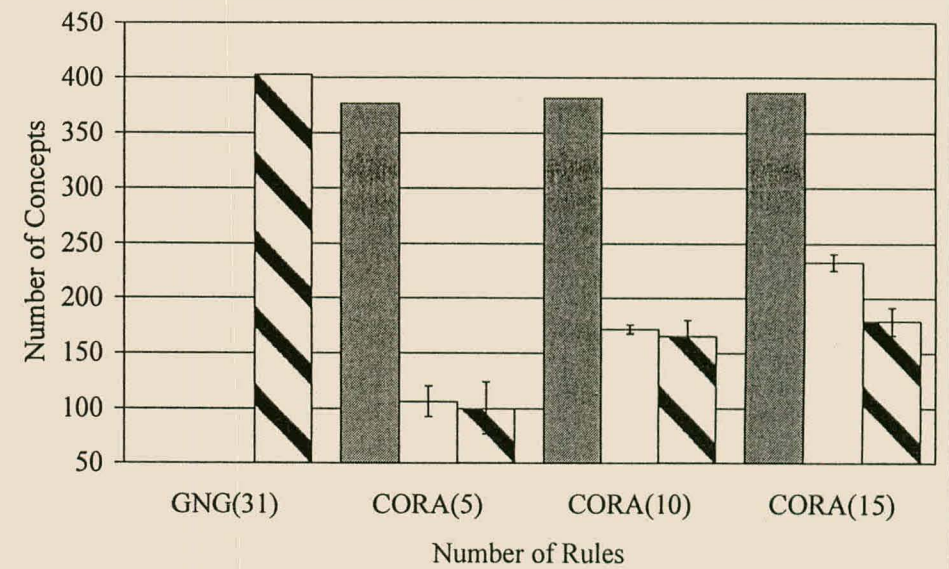
**Figure 6.33 Accuracy vs. Number of Rules for Housing Problem**



**Figure 6.34 Accuracy vs. Number of Rules for Servo Problem**

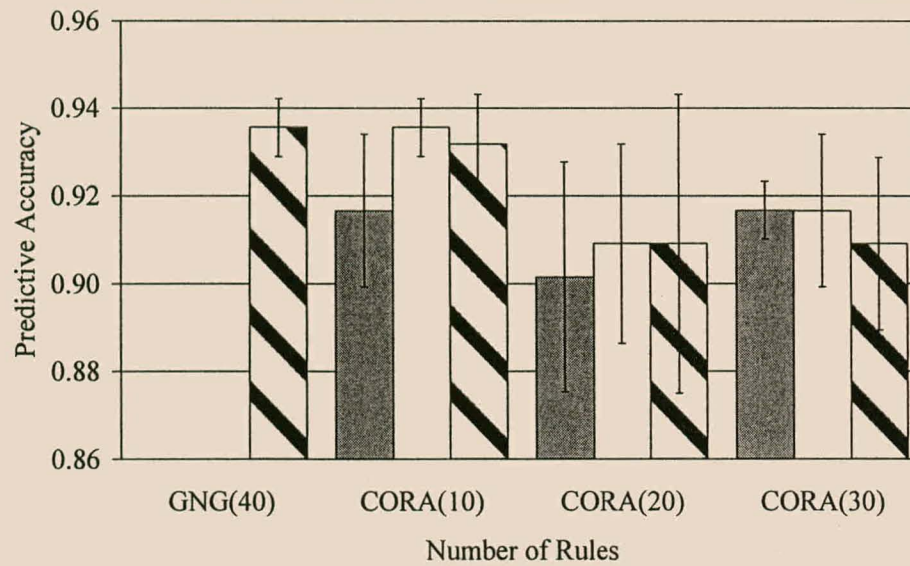


**Figure 6.35 No. of Concepts vs. No. of Rules for Housing Problem**

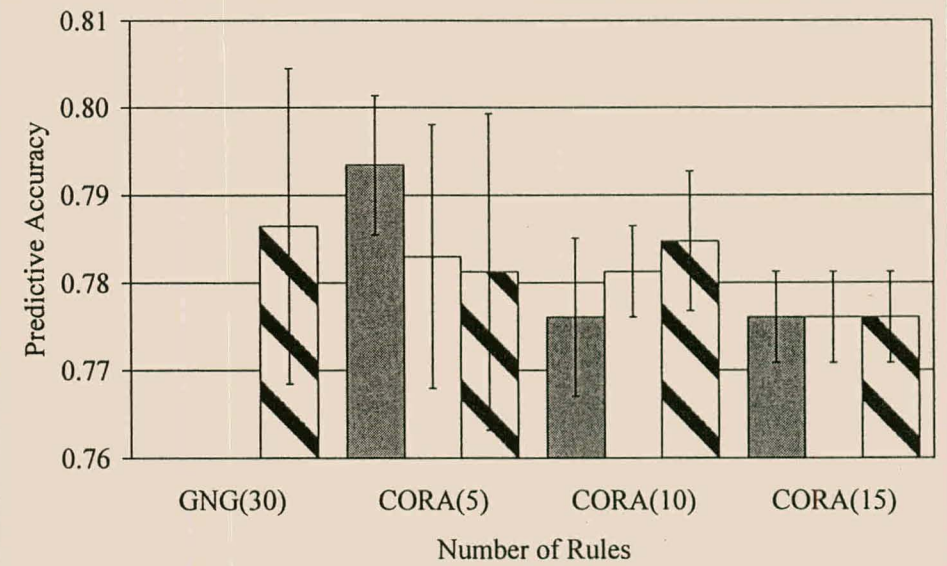


**Figure 6.36 No. of Concepts vs. No. of Rules for Servo Problem**

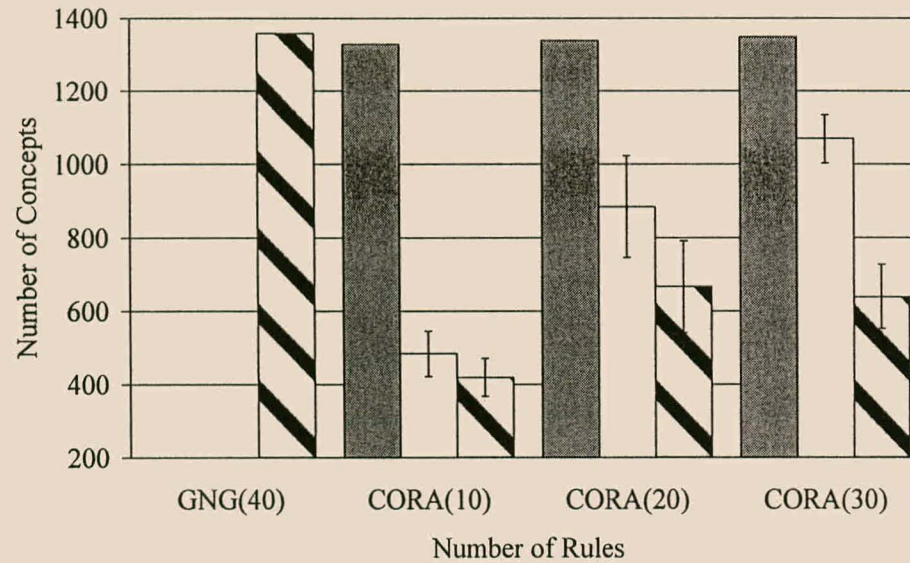




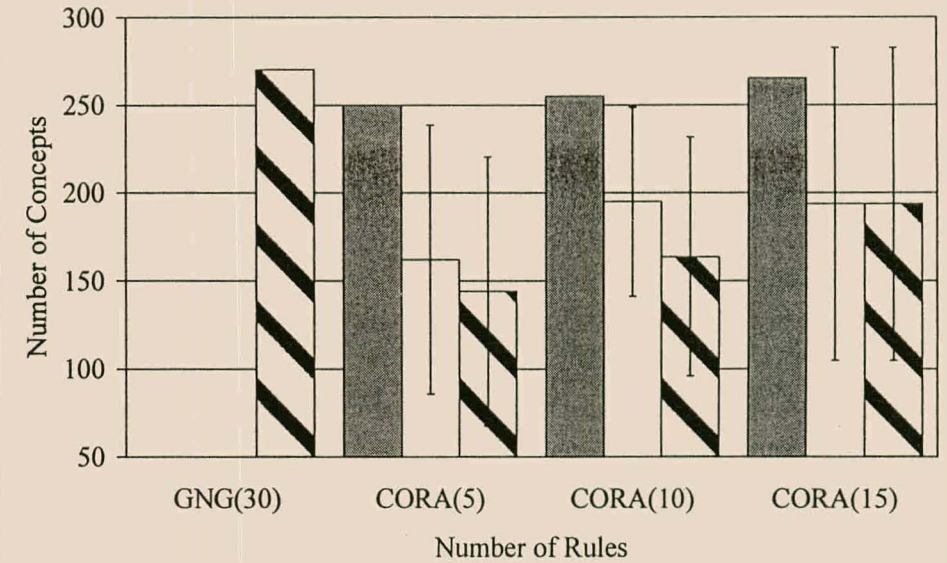
**Figure 6.37 Accuracy vs. Number of Rules for Ionosphere Problem**



**Figure 6.38 Accuracy vs. Number of Rules for Pima Problem**



**Figure 6.39 No. of Concepts vs. No. of Rules for Ionosphere Data**



**Figure 6.40 No. of Concepts vs. No. of Rules for Pima Problem**

The particular GNG and CORA algorithm training parameter settings used for the experiments in this section are summarised in table 6.1. Where a value for a particular training parameter is not specified the default value specified in table A.2 (for the GNG algorithm) or table A.3 (for the CORA algorithm) was used. Both table A.2 and A.3 can be found in appendix A.

Parameter	GNG Algorithm	CORA Algorithm
number_of_epochs	194   363   595   482   540   262   167	-
maximum_number_cell_nodes	See figures	-
number_of_iterations (RTS component)	-	2000   5000   1500   2000   500   5000   2500
number_of_rules	-	See figures
consequent_magnitude_penalty_factor	-	0.01   0.01   0.1   0.05   0.1   0.02   0.1
overlap_penalty_factor	-	0.5   0.1   0.1   0.5   0.1   0.1   0.5
minimum_swap_move_threshold	-	0.9
swap_move_ratio <sup>13</sup>	-	0.9

**Table 6.1    Parameter Settings for Number of Assembled Fuzzy Rules Experiments<sup>14</sup>**

The following observations and statements can be made regarding the influence of the number CORA-assembled fuzzy rules on model predictive accuracy and complexity:

- For all four of the regression problems (figures 6.29, 6.30, 6.33 and 6.34) the correct choice of the number of CORA-assembled fuzzy rules allowed the CORA algorithm to construct models with seemingly superior predictive accuracy in comparison to those generated by the GNG algorithm. This means that the ability to build internally disjunctive rules can provide one with an appreciably more powerful modelling language in comparison to those languages that do not encompass internally disjunctive rules. A similar conclusion was reached by Theron and Cloete (1996) who studied internal disjunction in terms of crisp “if...then...” rule induction.

<sup>13</sup> swap\_move\_ratio is the ratio of RTS iterations that use swaps to construct new solutions to those that use moves to perform the same task. For example, a ratio of 0.9 implies that for each 100 RTS search iterations in turn, the first ninety of these iterations use swaps to find new solutions and the last ten use moves.

<sup>14</sup> For the number\_of\_epochs, number\_of\_iterations, consequent\_magnitude\_penalty\_factor and overlap\_penalty\_factor training parameters, the specified values are for the Abalone, Auto, Housing, Servo, Ionosphere, Slugflow and Pima problems, respectively.

The results are not as clear for the two classification problems (figures 6.37 and 6.38). Neither the GNG nor the CORA algorithm consistently outperforms the other. In the author's opinion this is owing to the suboptimal choice of solution fitness function. The influence of this algorithmic property is discussed in more detail below.

Consider first the results after only the RTS component of the CORA algorithm had been used. For three of the four regression problems that were studied, an increase in the number of rules that could be assembled led to an increase in predictive accuracy. However, for the Auto (figure 6.30) and Servo (figure 6.34) problems the use of too many rules decreased the model predictive performance to levels below the optimum but above the worst results. For the remaining regression problem, the Abalone problem (figure 6.30), an increase in the number of assembled rules led to a slow decline in predictive performance. The results for the two classification problems are not as clear as those for the regression problems but an increase in the number of assembled rules generally led to a decrease or at best to no change in predictive performance.

It seems therefore, for regression problems in particular, that an increase in the number of assembled fuzzy rules is generally beneficial in terms of predictive accuracy. However, in some cases too many rules seem to have provided the CORA algorithm with too much modelling power. In other words the CORA algorithm constructs models that overfit on the training data<sup>15</sup> and in turn generalise poorly on the validation data.

In the case of classification problems the above phenomenon is coupled with the disadvantages of using a regression-based objective function to evaluate solutions instead of a classification-based one. In other words, the generally, but not consistently downward trend in predictive performance as the number of assembled rules is increased can be attributed to overfitting as well as to the use of a suboptimal objective function. This observation is supported by the relatively high standard deviation of the classification results after the RTS component has finished execution.

---

<sup>15</sup> All CORA models for a particular problem were generated using the same number of RTS iterations.

- Changes to the number of assembled fuzzy rules do not seem to have caused a clear trend in the effect of membership function merging. Nevertheless, merging has for most problems a negative effect on predictive performance. The exceptions are the Auto and Ionosphere problems. For these two problems the average model predictive performance increases or does not change if membership function merging is performed, although the high standard deviation of the results makes it difficult to determine whether these trends are significant.

However, if one considers the “number of concepts” figures 6.31, 6.32, 6.35, 6.36, 6.39 and 6.40, the average decrease in predictive accuracy for some problems is, in the author’s opinion, outweighed by the appreciable gain in model simplicity. Consider table 6.2 below.

Problem	Relevant Figures	Predictive Accuracy Decrease (%)	Model Complexity Decrease <sup>16</sup> (%)
Abalone	6.29, 6.31 and B.1	1.2	17   15
Housing	6.33, 6.35 and B.3	2.3	41   38
Servo	6.34, 6.36 and B.4	18	72   83
Pima	6.38, 6.40 and B.6	1.3	35   40

**Table 6.2 Average Decrease in Predictive Accuracy and Model Complexity Owing to Membership Function Merging**

The table lists the problems for which membership function merging caused a decrease in model predictive accuracy. For each problem the case (number of fuzzy rules) is shown where predictive accuracy decreased the most. For the Abalone problem this occurred when ten rules were assembled, for the Housing problem when twenty rules were used, the Servo problem when five rules were constructed and for the Pima problem this occurred when five rules were assembled.

Table 6.2 shows that for three (Abalone, Housing, Pima) of the four problems where membership function merging had a detrimental effect on predictive performance the average decrease in accuracy was 1.6%. For these problems model concept complexity on average dropped by 31%. Unfortunately, for the Abalone, Housing

---

<sup>16</sup> The last column of table 6.2 contains two numbers for each problem. The first number is the decrease in model concept complexity and the second number is the drop in parameter complexity. Percentages are calculated based on the largest model complexity value, i.e. before merging took place.



and Pima problems in particular, the high standard deviation of the results make it difficult to determine whether the drop in model complexities is always significant.

The detrimental effect of membership function merging is most pronounced for the Servo problem. This can be attributed to the fact that the Servo problem consists of few data and contains 83% discrete or binary attributes (upon presentation of the data to the GNG and CORA algorithms). As discussed in section 6.3.1.2 these two factors cause the CORA algorithm to model the Servo data relatively poorly. Apart from overfitting on the training data, the RTS component of the CORA algorithm seems to use combinations of the uppermost sections of membership function Gaussian curves to create decision boundaries for problems consisting primarily of discrete attributes. The method of membership function merging used by the MRG component smoothes these uppermost curve sections (see section 4.3.4.4 and figure 4.18 for details). Such smoothing seems to disrupt the RTS-created model decision boundaries to such a degree that predictive performance is seemingly significantly harmed.

- In contrast to membership function merging, the effect of the RED component does exhibit a relationship with respect to changes in the number of assembled fuzzy rules. In particular, as the number of constructed fuzzy rules increases, so the level of complexity reduction increases (see figures 6.31, 6.32, 6.35, 6.36, 6.39, 6.40 as well as figures B.1 through B.6). The complexity reduction seems substantial, especially if the results are considered for the experiments where the greatest number of fuzzy rules was utilised. A summary of these particular results is presented in table 6.3.

Problem	Model Complexity Decrease <sup>17</sup> (%)	
Abalone	8.1	8.0
Auto	15	14
Housing	30	30
Servo	23	23
Ionosphere	40	40
Pima	0	16

**Table 6.3    Average Model Complexity Decrease as a Result of Rule Reduction**

<sup>17</sup> Percentages are calculated based on the largest model complexity value, in other words after merging took place. The first number given in the column is the percentage decrease in model concept complexity and the second number is the drop in parameter complexity.



Unfortunately the significance of the reduction in complexity is for some problems difficult to determine owing to the high standard deviation of the results.

Rule reduction does not seem to be linked with a corresponding change (increase or decrease) in predictive accuracy. In other words, the trend of increased complexity reduction does not seem to be linked with any seemingly consistent change in model predictive accuracy. In some cases rule reduction seems to on average improve predictive performance, e.g. the Abalone problem (using thirty rules), the Housing problem (thirty rules) and the Pima problem (ten rules).

The worst case of where accuracy is negatively influenced is the Ionosphere problem (figure 6.37). Here a drop in  $R^2$  of 0.917 to 0.909, or 0.87%, in accuracy is found where thirty fuzzy rules are used. (Again, these results should be treated with caution. owing to the high sample standard deviation.) The reason why the worst results are obtained with a classification problem is most likely because the RED component decides which rules to ignore based on a  $R^2$  accuracy measure, rather than a classification accuracy measure. In other words classification performance is not explicitly considered.

## 6.2.2 Percentage of RTS Swaps or Moves Evaluated per Iteration

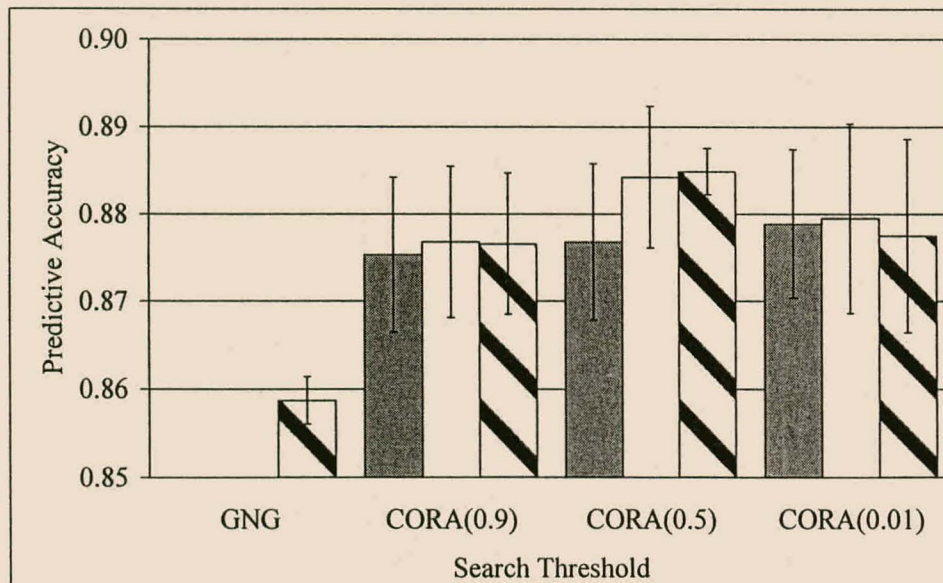
This section presents results of experiments that were performed to study the effect of the RTS minimum swap or move threshold<sup>18</sup> on model predictive accuracy and complexity. Three levels of the threshold were studied, viz. using at most 10%, 50% or 99% of available swaps or moves. All other training parameters were set to the values given in table 6.1. Where a value for a particular training parameter is not specified in the table the default value specified in table A.3 in appendix A was used.

---

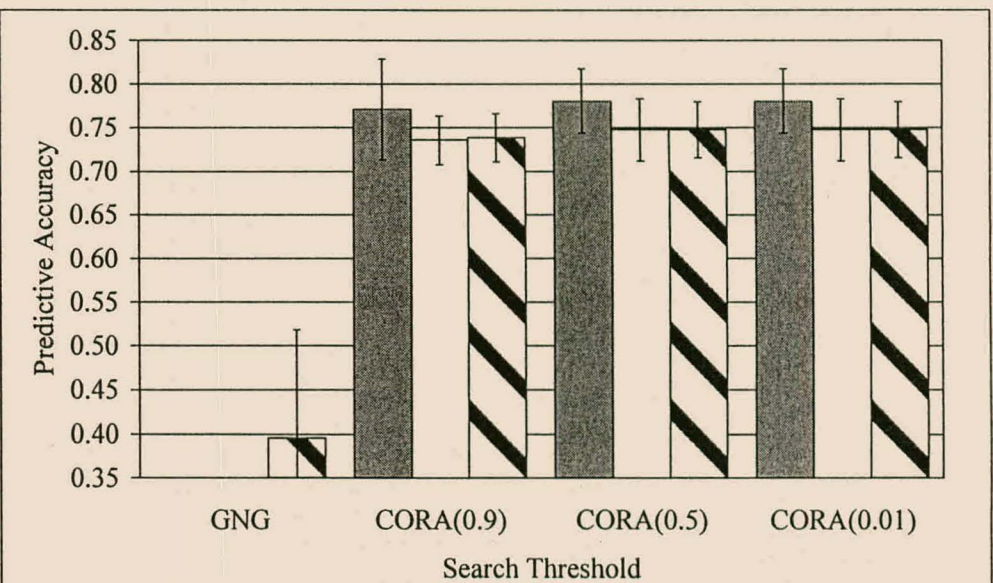
<sup>18</sup> The term “minimum swap (or move) threshold” is defined as the maximum percentage of all possible and permissible solutions (generated by applying a single swap or move to the current solution) that may be evaluated during any RTS combinatorial search iteration. A threshold of 0.9 means that only 10% of all possible and permissible solutions may be evaluated. As another example, a threshold of 0.01 means that 99% of all possible and permissible swaps or moves may be evaluated. Furthermore, the term “search resolution” refers to the current swap (or move) threshold used by the RTS algorithm (see section 4.3.4.9 for details on how this current threshold is calculated). These definitions of minimum and current swap (or move) threshold are identical to those used elsewhere in this dissertation.

The problems considered in this section are the Auto, Servo, Slugflow and Pima problems. The Abalone, Housing and Ionosphere problems were not studied, because experiments that allowed the consideration of more than 10% of available swaps or moves during the RTS combinatorial search took too long to perform, given the time constraints set for this investigation. In addition, only one `number_of_rules` (number of CORA rules to construct) setting was tested for the Slugflow problem because the GNG algorithm generated a model with so few rules. Finally, it is important to note that for a particular problem all experiments were run for the same number of RTS search iterations, no matter what level of RTS minimum swap threshold was employed.

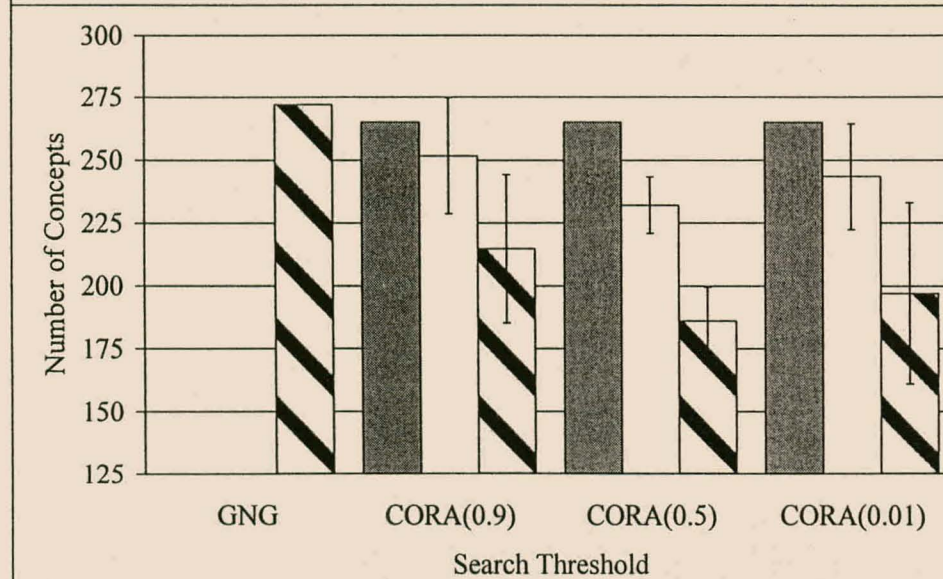
Typical experimental results are presented in figures 6.41 through 6.48. These figures show how rule model predictive accuracy as well as the number of model concepts is affected by changes in RTS search resolution. Only the results are shown of experiments where the maximum number of CORA fuzzy rules was used. The remainder of the results are presented in appendix B, specifically figures B.25 through B.46. The configuration of each figure is similar to that of the figures used in section 6.3.2.1. The only difference (apart from the actual results) is that the number in parentheses after the CORA labels indicates the search resolution rather than the number of fuzzy rules.



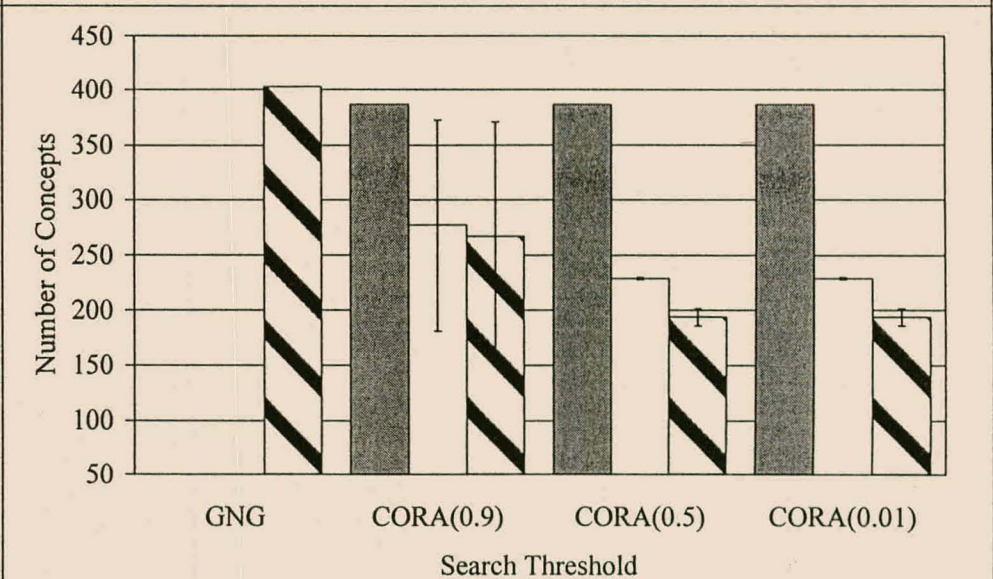
**Figure 6.41 Accuracy vs. Search Resolution for Auto Data**



**Figure 6.42 Accuracy vs. Search Resolution for Servo Data**

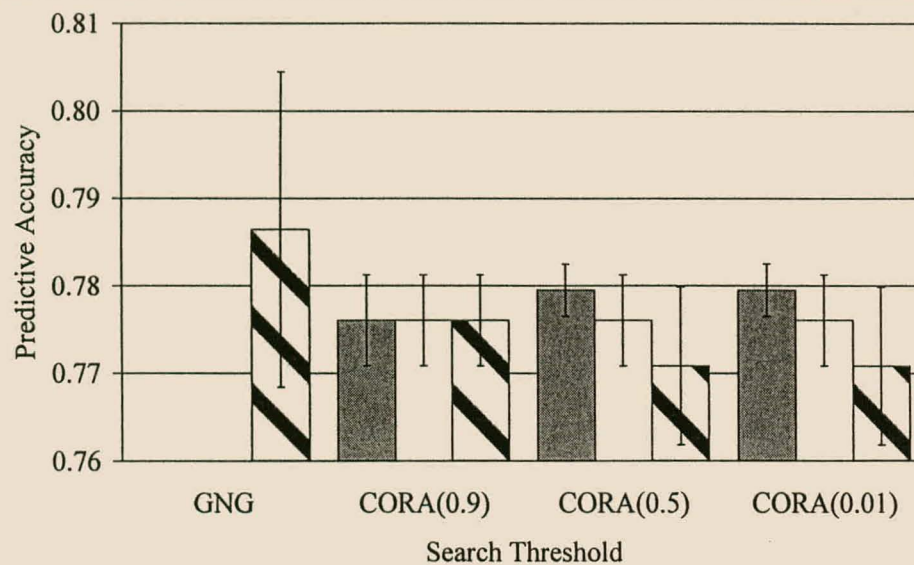


**Figure 6.43 No. of Concepts vs. Search Resolution for Auto Data**

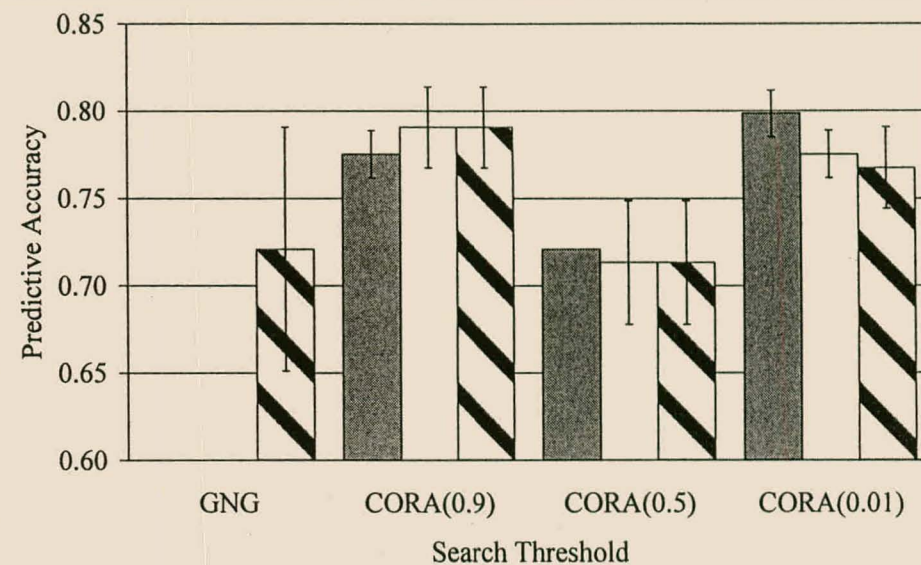


**Figure 6.44 No. of Concepts vs. Search Resolution for Servo Data**

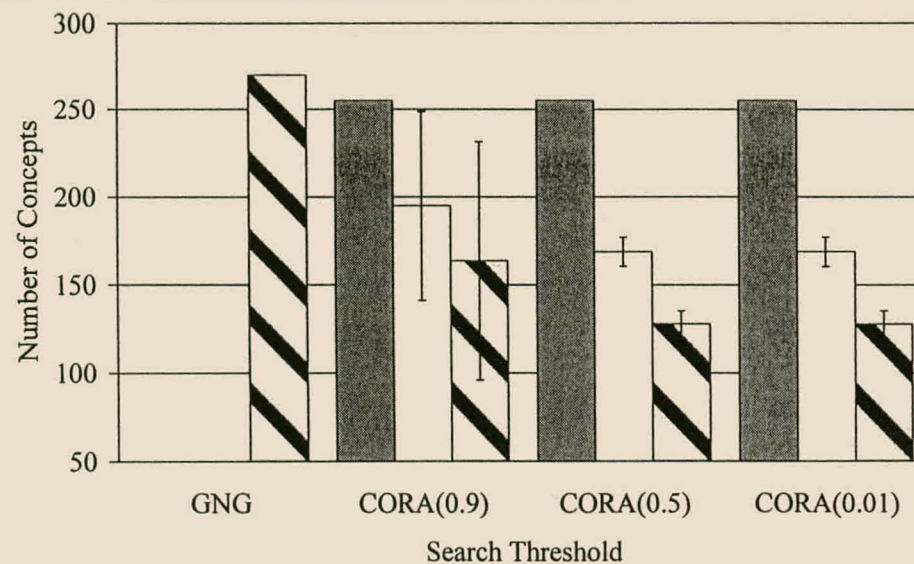




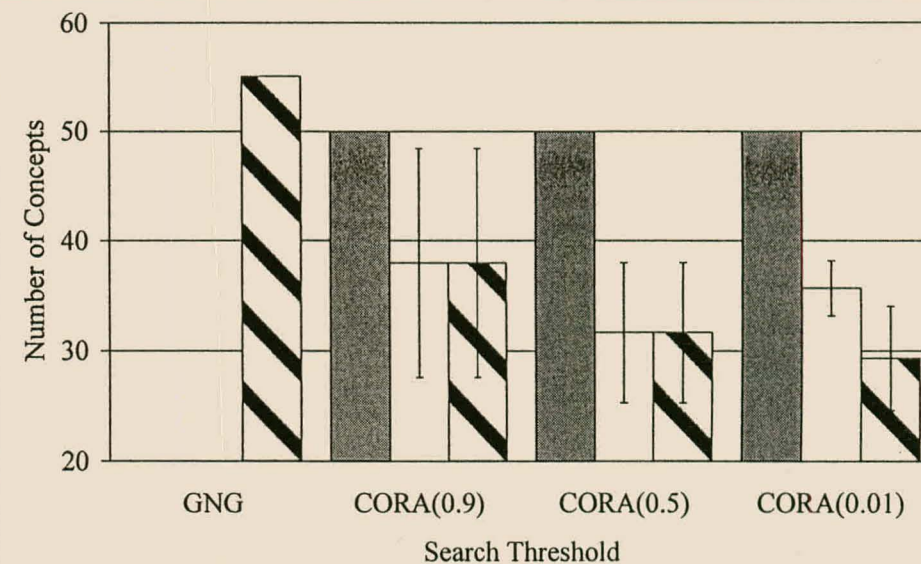
**Figure 6.45 Accuracy vs. Search Resolution for Pima Data**



**Figure 6.46 Accuracy vs. Search Resolution for Slugflow Data**



**Figure 6.47 No. of Concepts vs. Search Resolution for Pima Data**



**Figure 6.48 No. of Concepts vs. Search Resolution for Slugflow Data**

Inspection and analysis of the above-mentioned figures reveal the following with regard to the influence of changes in RTS search resolution on CORA rule model predictive accuracy and complexity:

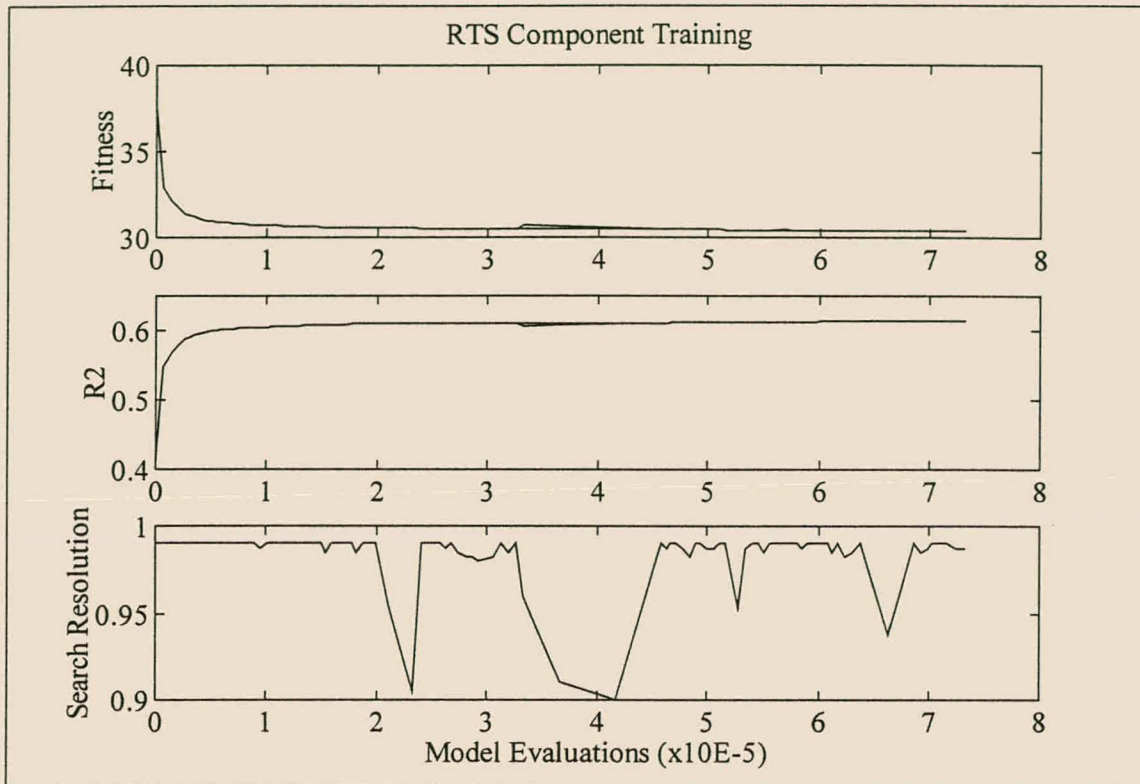
- Consider first the accuracy results obtained by the RTS component of the CORA algorithm. In general there does not seem to be a clear relationship between the minimum swap (or move) threshold and rule model predictive accuracy. Improvements in accuracy as a result of a less stringent swap (or move) threshold are at most 2.9% (for the Slugflow problem) and at worst 3.3% (for the Servo problem using five rules). In addition, the results for most experiments exhibit large standard deviations. From this can be concluded that a decrease in RTS swap (or move) threshold (i.e. more swaps or moves are tested) generally does not lead to an increase in model predictive accuracy.
- Next, consider the predictive accuracy results obtained after the MRG and RED components of the CORA algorithm have been used. As with the RTS component results, no generally valid trends can be observed as the RTS minimum swap (or move) threshold changes. Likewise, inspection of figures 6.43, 6.44 and 6.47, as well as figures B.33, B.34, B.35, B.36, B.42 and B.44, reveals that changes in minimum threshold do not seem to have any significant effect on the level of model simplification that is performed by the MRG or RED components.

In summary, no significant change in either predictive accuracy or model complexity reduction with respect to changes in swap (or move) threshold can be observed in the experimental results for this section. On the other hand, this means that substantial reductions in RTS computation time can be obtained without appreciably affecting final trained model accuracy or complexity. For example, if the minimum swap threshold is set at 0.9 then at worst (the RTS search never finds a better solution than the initial one) the RTS computational time will only be 10% of what it would be without the minimum swap threshold constraint.

However, as described in section 4.3.4.9, it must be kept in mind that all RTS combinatorial searches used an initial search resolution of one percent. Inspection of trends in RTS training accuracy brought to light that for most problems the RTS search resolution did not constantly or consistently deviate from one percent during the initial rule assembly phase. Figure 6.49 shows typical training fitness and accuracy (in terms of  $R^2$ ) curves together with the corresponding search resolution curve. As shown in figure 6.49 this initial training phase is usually where most



improvements in model fitness are made by the RTS algorithm. This phenomenon is especially true if many membership functions are being apportioned amongst many rules. In this case the RTS component usually found it relatively easy to discover new rule models with improved fitness.



**Figure 6.49 Training Accuracy and RTS Search Resolution against RTS Iterations<sup>19</sup>**

In the light of this, it seems that the adaptive modification of the search resolution by the RTS component, from an initial value of one percent up to the swap (or move) threshold, plays a more significant role in determining final rule model properties than the minimum swap (or move) threshold does. In particular, the use of an initial search resolution of one percent in all experiments that use the same set of parameters, with the exception of the minimum swap (or move) threshold, most likely causes the RTS search trajectory in each of these experiments to be

<sup>19</sup> The results given in figure 6.49 are for an experiment based on the Abalone problem. Ten fuzzy rules were assembled. The minimum swap (or move) threshold was set at 0.9. The swap to move ratio was also set to 0.9. The random seed was 5. Other training parameters were set to default values (see table A.3 in appendix A).

The first two subplots in figure 6.49 each present two curves, viz. the best fitness (or accuracy) and the current fitness (or accuracy). By and large these plots are on top of each other, except between roughly 330000 and 450000 RTS iterations.

roughly the same. This in turn will mean that the trained models will be more or less the same. This hypothesis is substantiated by the lack of any significant trends in the results presented for this section.

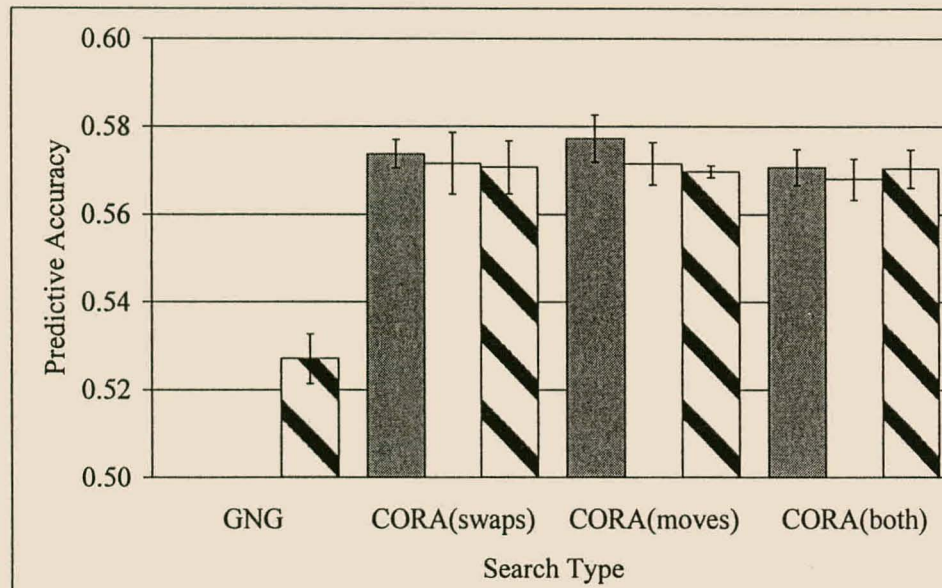
### 6.2.3 Rule Construction Using Swaps, Moves or Both

The next aspect of the CORA algorithm that will be discussed concerns the effects of using swaps, moves or both swaps and moves during the RTS combinatorial search. The reader is referred to section 4.3.4.9 for more details regarding the difference between swaps and moves, as well as how the RTS algorithm alternates between swaps and moves (if this training option is chosen).

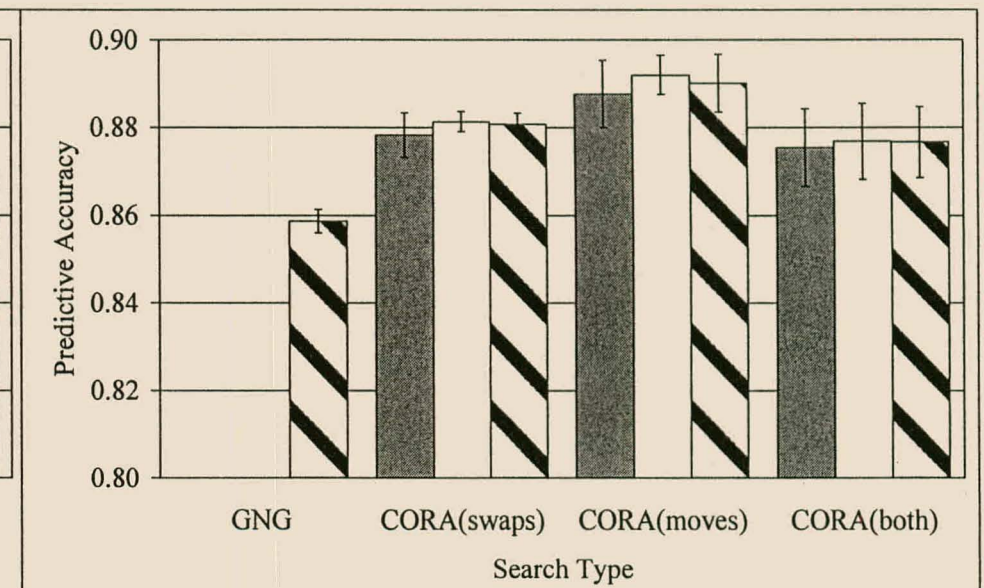
The problems that are studied are the Abalone, Auto, Housing, Servo, Ionosphere, Slugflow and Pima problems. The influence of using swaps, moves or both of these is evaluated in terms of rule model predictive accuracy and complexity. For each problem the use of swaps, moves or both is evaluated for each of the `number_of_rules` settings studied in section 6.3.2.1. If both swaps and moves are employed, the `swap_move_ratio` is set to 0.9. The reason for this choice of ratio is because it was originally felt that moves should be used to make the RTS component capable of changing the initial membership function distribution presented to the RTS component, rather than being the primary means of generating new solutions. If swaps are used the membership function distribution will stay the same throughout a RTS combinatorial search.

Otherwise, the same training parameter settings as those given in table 6.1 were employed. Predictive accuracy and concept complexity results are presented in figures 6.50 through 6.63. Additional accuracy, concept complexity, as well as parameter complexity results are presented in appendix B, specifically figures B.47 through B.89. In each figure caption, the abbreviation “SMB” stands for “swaps, moves or both swaps and moves”. The number in parentheses at the end of each figure caption is the number of rules used to construct the relevant rule models.

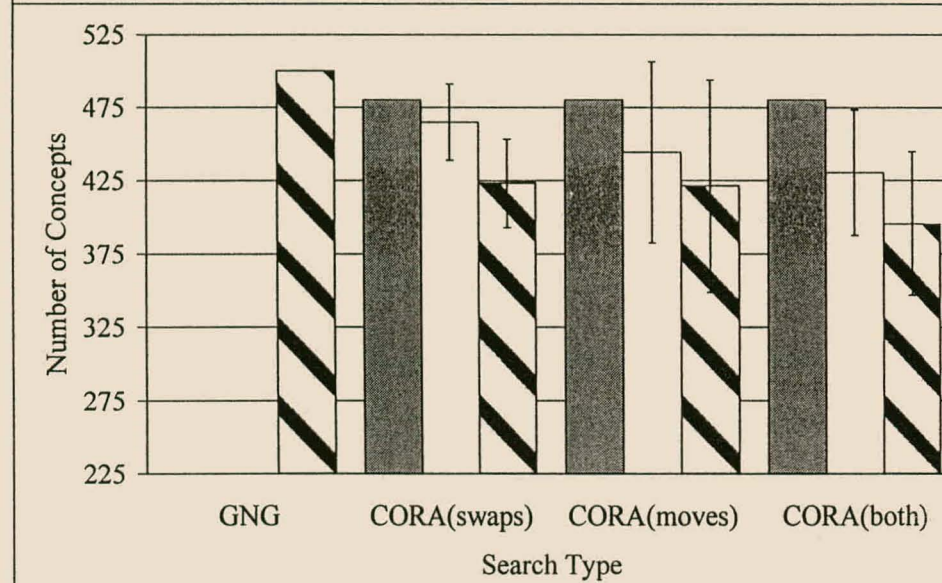




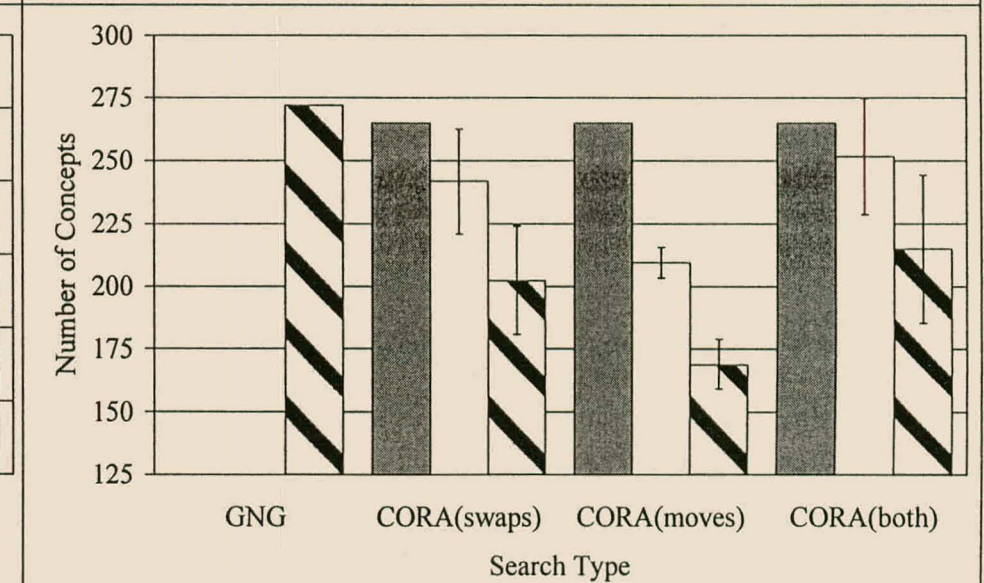
**Figure 6.50 Accuracy vs. SMB for Abalone Problem (30 rules)**



**Figure 6.51 Accuracy vs. SMB for Auto Problem (27 rules)**

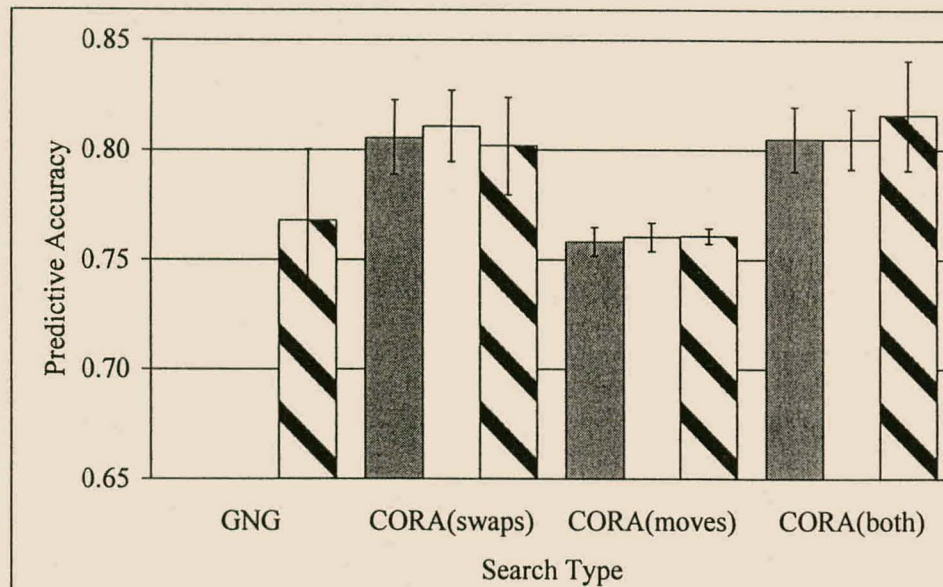


**Figure 6.52 No. of Concepts vs. SMB for Abalone Data (30 rules)**

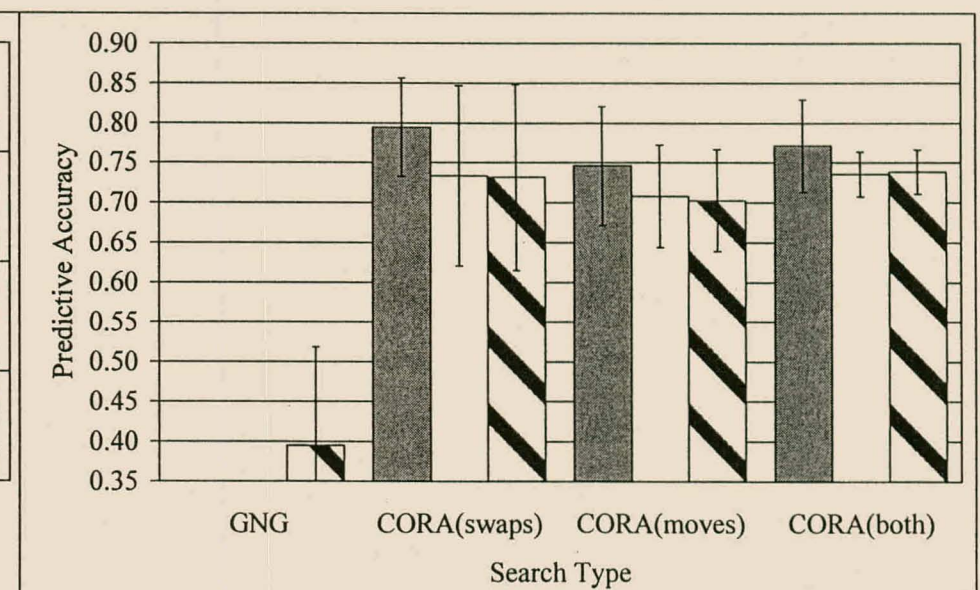


**Figure 6.53 No. of Concepts vs. SMB for Auto Problem (27 rules)**

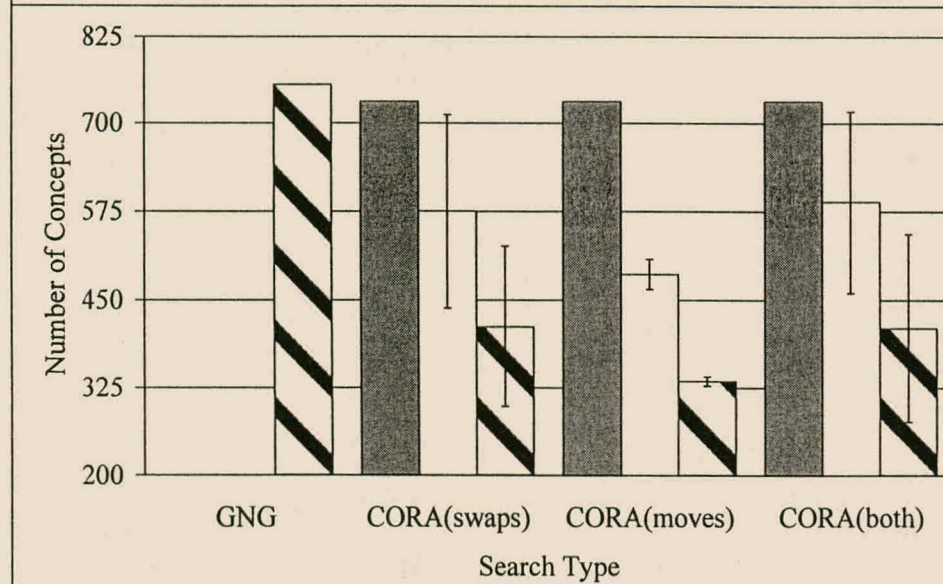




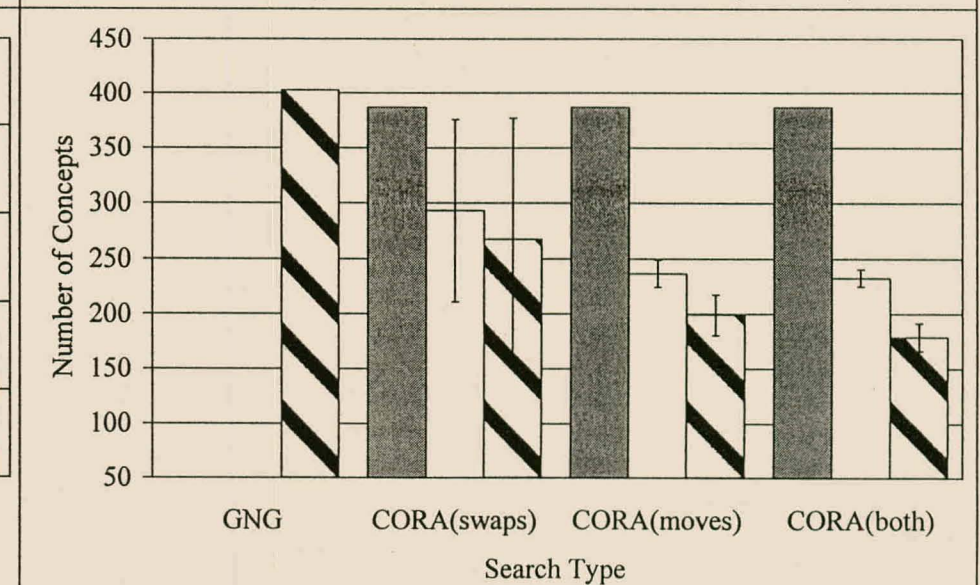
**Figure 6.54 Accuracy vs. SMB for Housing Problem (30 rules)**



**Figure 6.55 Accuracy vs. SMB for Servo Problem (15 rules)**

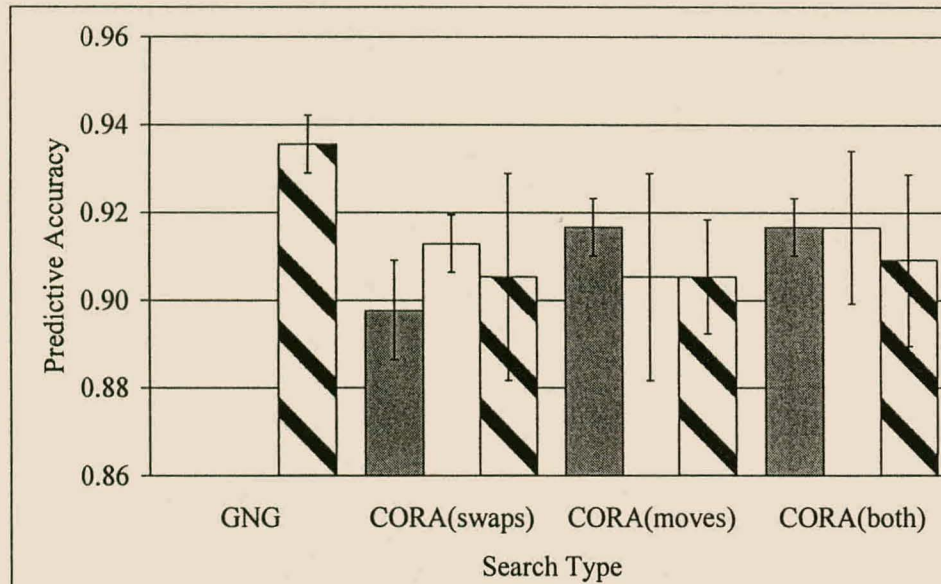


**Figure 6.56 Concepts vs. SMB for Housing Problem (30 rules)**

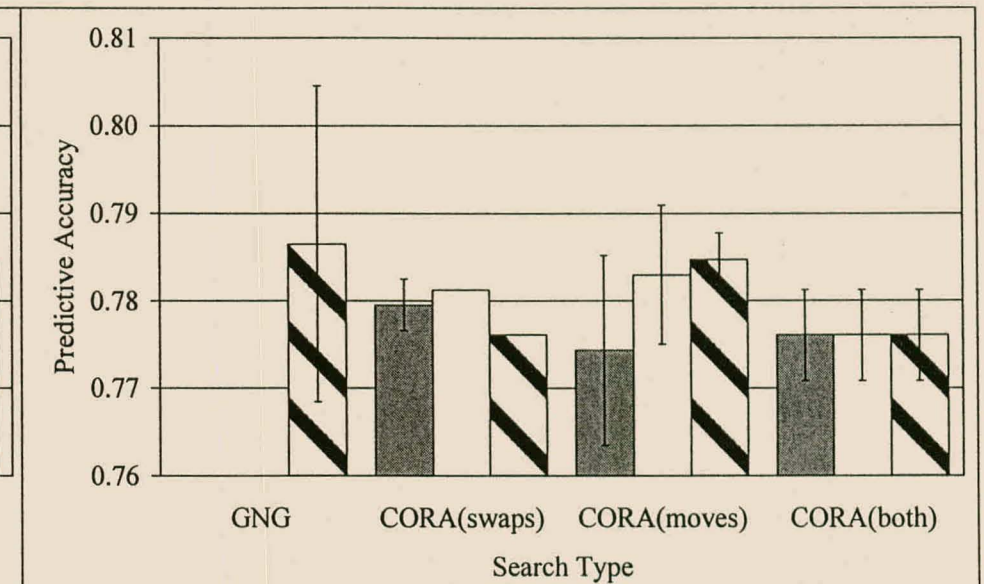


**Figure 6.57 No. of Concepts vs. SMB for Servo Data (15 rules)**

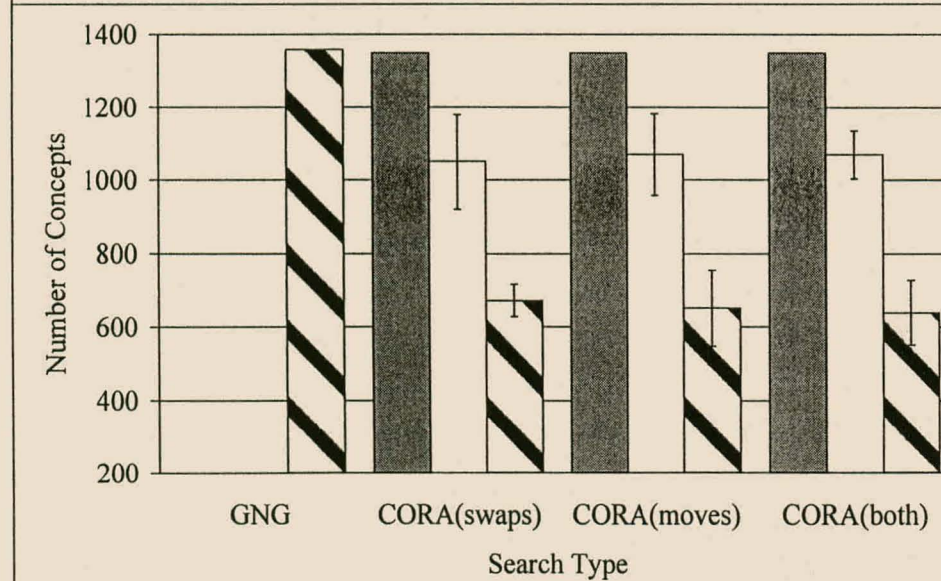




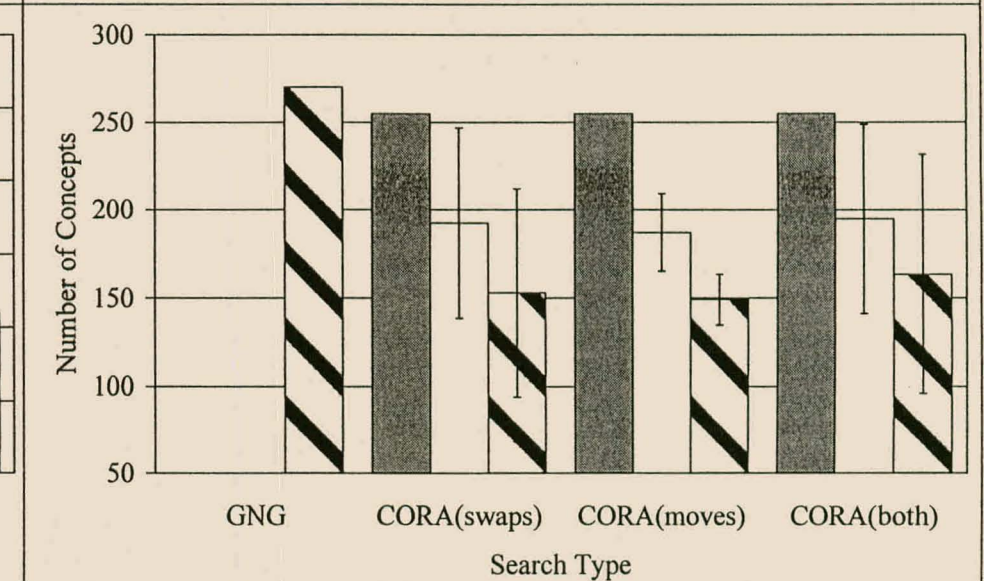
**Figure 6.58 Accuracy vs. SMB for Ionosphere Problem (30 rules)**



**Figure 6.59 Accuracy vs. SMB for Pima Problem (15 rules)**

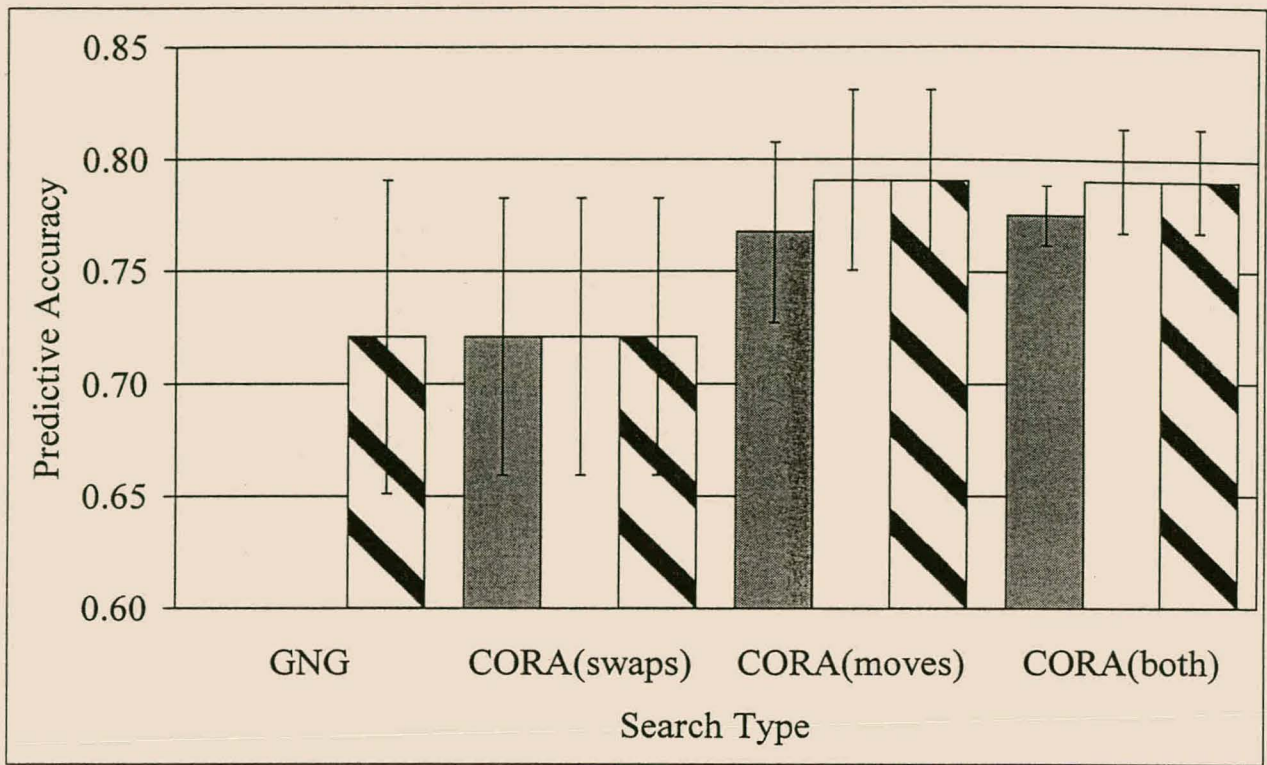


**Figure 6.60 Concepts vs. SMB for Ionosphere Problem (30 rules)**

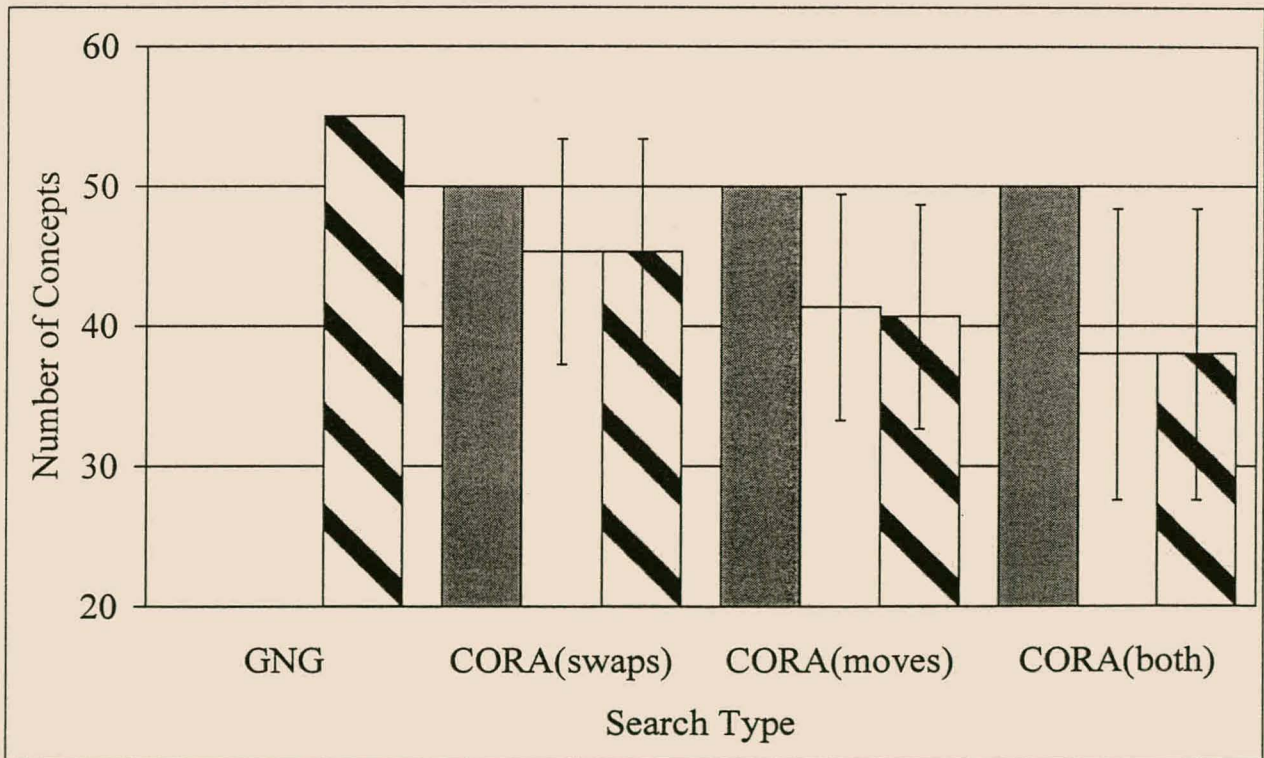


**Figure 6.61 Concepts vs. SMB for Pima Problem (15 rules)**





**Figure 6.62 Accuracy vs. SMB for Slugflow Problem (6 rules)**



**Figure 6.63 No. of Concepts vs. SMB for Slugflow Data (6 rules)**

The following observations can be made regarding the figures presented in this section as well as those presented in appendix B.

- Consider first the effect on predictive accuracy. Note that for this discussion the results obtained using swaps exclusively are assumed to be the standard against which the results obtained using the other two training variants (moves only or both swaps and moves) are measured.

The exclusive use of moves seemingly improved predictive accuracy for the Auto, Ionosphere and Slugflow problems (see figures 6.51, 6.58 and 6.62) when the maximum number of rules was assembled. In the last case the high standard deviation of the results makes it difficult to determine whether the increase in accuracy is significant. However, if fewer rules are assembled and moves are used exclusively for the Auto and Ionosphere problems, or if any number of rules is assembled using both swaps and moves, there seems to have been no substantial change in predictive accuracy.

For the Abalone, Servo and Pima problems the use of either moves or both swaps and moves together with any number of assembled rules seems to have no appreciable effect on accuracy. The one exception is where ten rules were assembled for the Servo problem. In this case the exclusive use of moves resulted in appreciably worse predictive accuracy.

Significantly worse results were obtained for the Housing problem if the maximum allowed thirty rules were assembled using moves (see figure 6.54). Likewise, if fewer rules are assembled (see figures B.66 and B.68) the predictive performance of the derived rule models drops, although the high standard deviation of the results makes it difficult to determine if this drop is significant.

- In general the use of moves seems to have allowed greater model complexity reduction than using swaps did. However, in only three cases, viz. the Auto problem using 27 rules (figure 6.53), the Abalone problem using ten rules (figure B.60) and the Ionosphere problem using ten rules (figure B.84), did the use of moves lead to seemingly substantial reductions in model complexity. For the other experiments the high variation in the results makes it difficult to determine whether the drop in model complexity is significant.

In summary, the use of either moves or both swaps and moves (based on a `swap_move_ratio` of 0.9) instead of swaps only, does not seem to have had a consistently beneficial or detrimental effect on model predictive accuracy. In terms of model complexity, the use of moves instead of swaps generally led to greater complexity reduction by the MRG and RED components of the CORA algorithm. It is hypothesised that the reason for these phenomena is the following. As described in section 4.3.4.9, a move consists of passing a single membership function from one fuzzy rule to another. No reciprocal membership function moves is performed from the latter rule to the former. The only restriction that is placed on this move is that it must meet the permissibility requirements set out in section 4.3.1.2 with respect to the membership function that is being moved and the rule that it is moved to.

In contrast, the effect of the move on the rule that loses the membership function is ignored. In other words, any membership function overlapping that has been constructed by the RTS component of the CORA algorithm is not taken into account. Such creation of membership function overlapping in one rule and possible destruction of overlapping in the other rule can lead to a set of fuzzy rules with two properties, one negative and the other positive. The negative effect is that the rule model overfits on the training data and thus generalises poorly. In other words, owing to the relatively relaxed restrictions placed on the combinatorial search when using moves, the RTS component is more able, than when swaps are used, to find some model that fits the training data well rather than the underlying model represented by the data. The advantage of using moves instead of swaps is that in general more rule model simplification can be performed. Owing to the more relaxed permissibility restrictions the RTS component is more able to create fuzzy rules with antecedent parts that exhibit a greater degree of overlapping. This in turn allows more membership function merging to take place.

## 6.2.4 Consequent Magnitude Penalisation

This section discusses the results obtained for the experiments performed to investigate the effect on model predictive accuracy and complexity of consequent magnitude penalisation. The problems considered in this section are the Abalone, Auto, Housing, Servo, Slugflow and Pima problems. The results obtained for these problems are presented in figures 6.64 through 6.75 as well as the figures in appendix B, specifically figures B.90 through B.95. The former set of figures give the accuracy and concept complexity results and the latter set of figures present the parameter complexity results.

Each figure gives the results for a particular problem, for all the `number_of_rules` settings employed in section 6.3.2.1. All other CORA training parameter settings were identical to those given in table 6.1, with the exception of the consequent magnitude penalisation factor (set to the value in table 6.1 or set to zero). Both swaps and moves were employed based upon the `swap_move_ratio` given in table 6.1. In each figure the horizontal axis label for a particular subset of experimental runs gives the number of rules assembled in parentheses and whether consequent magnitude penalisation was employed or not (Yes or No).

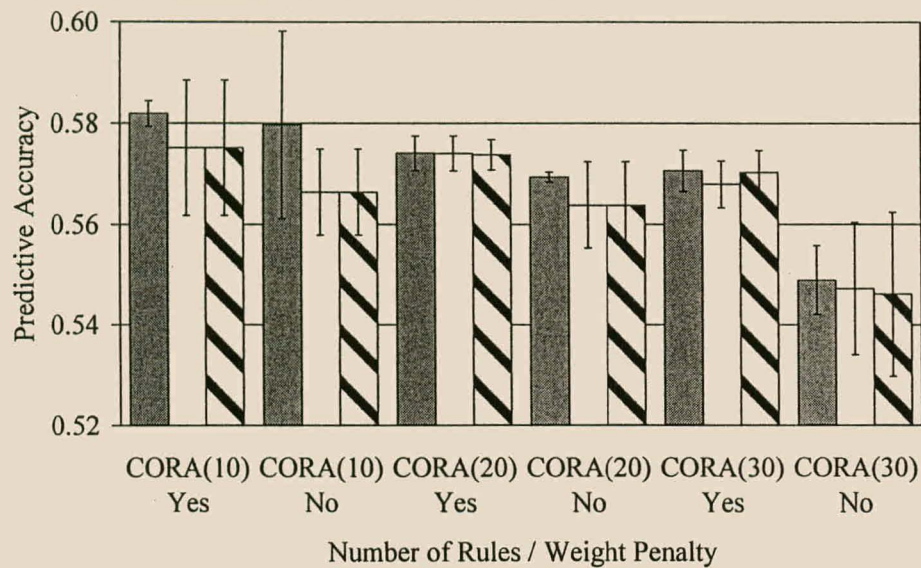
Inspection of the above figures as well as those in appendix B, specifically B.90 through B.95, reveals the following:

- Consequent magnitude penalisation generally improves rule model predictive accuracy in comparison to the situation where no penalisation is performed. This is especially the case if, for a particular problem, relatively many rules are assembled. The single exception to these phenomena is the results for the Servo problem (see figure 6.69). For this problem the use of consequent magnitude penalisation has a substantially negative effect on predictive accuracy, no matter how many fuzzy rules are assembled.

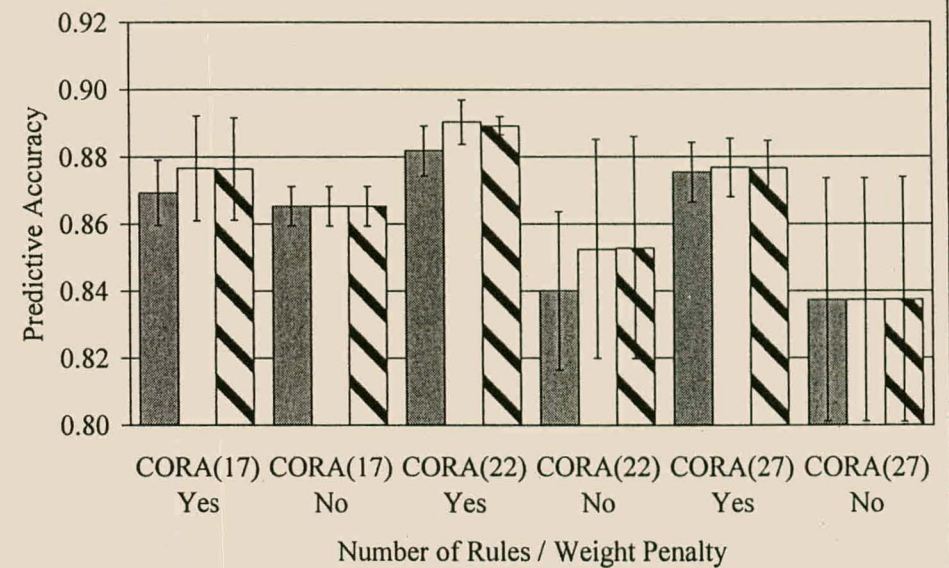
The general improvement in predictive accuracy can be attributed to the fact that the use of consequent magnitude penalisation forces the RTS combinatorial search to look for rule models that fit the data well but that also generate a relatively smoother output prediction surface. Models that generate a more smooth output surface usually generalise better than those that generate a rougher surface do, for the same training accuracy.

In addition, consequent magnitude penalisation leads to rules with small consequents. In other words, small changes in the antecedent part and thus the predicted output of a given fuzzy rule (caused by membership function merging) should lead to small changes in the overall fuzzy rule model output and therefore not influence model accuracy much. In contrast, small changes in antecedent part of a fuzzy rule with a large consequent can cause large changes, usually detrimental, in the fuzzy rule output and in turn the fuzzy rule model output.

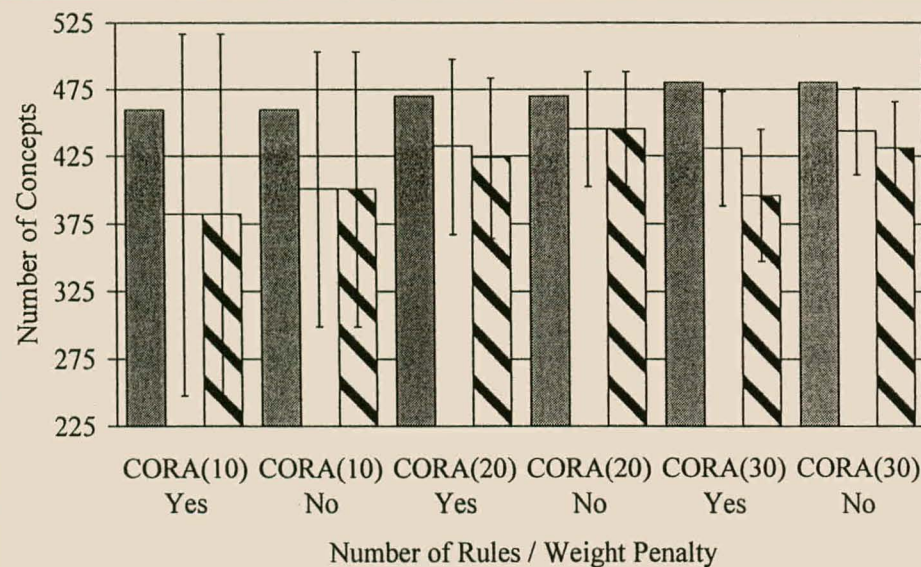




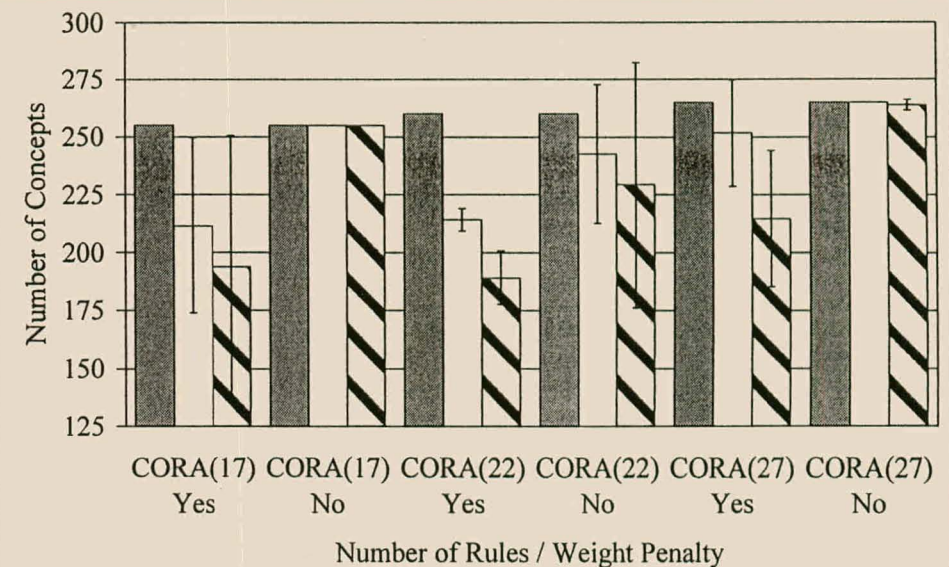
**Figure 6.64 Accuracy vs. Use of Weight Penalty for Abalone Data**



**Figure 6.65 Accuracy vs. Use of Weight Penalty for Auto Problem**

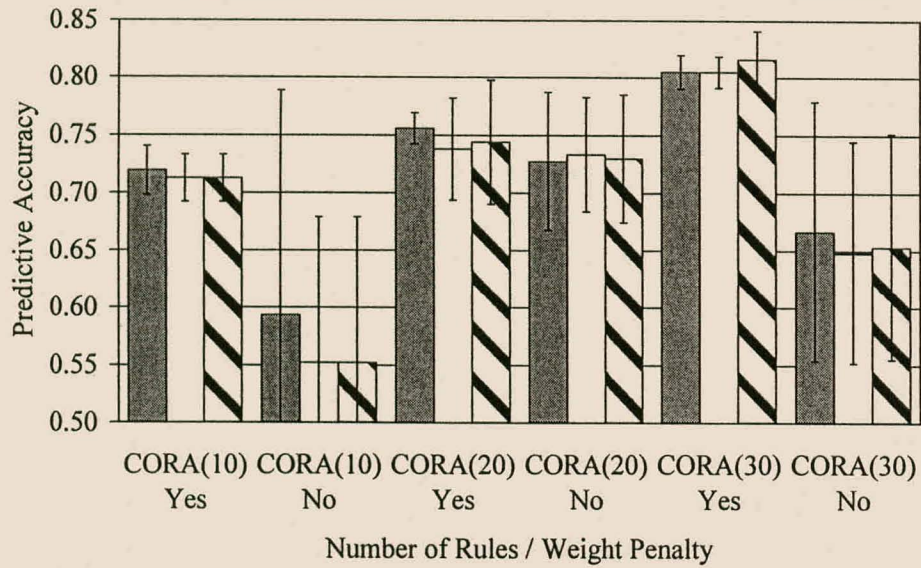


**Figure 6.66 Concepts vs. Use of Weight Penalty for Abalone Data**

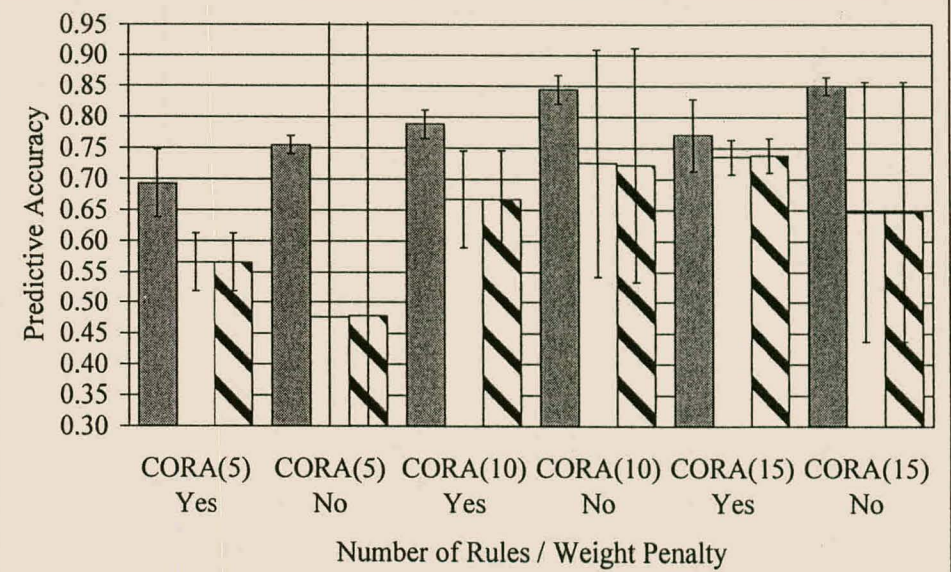


**Figure 6.67 Concepts vs. Use of Weight Penalty for Auto Problem**

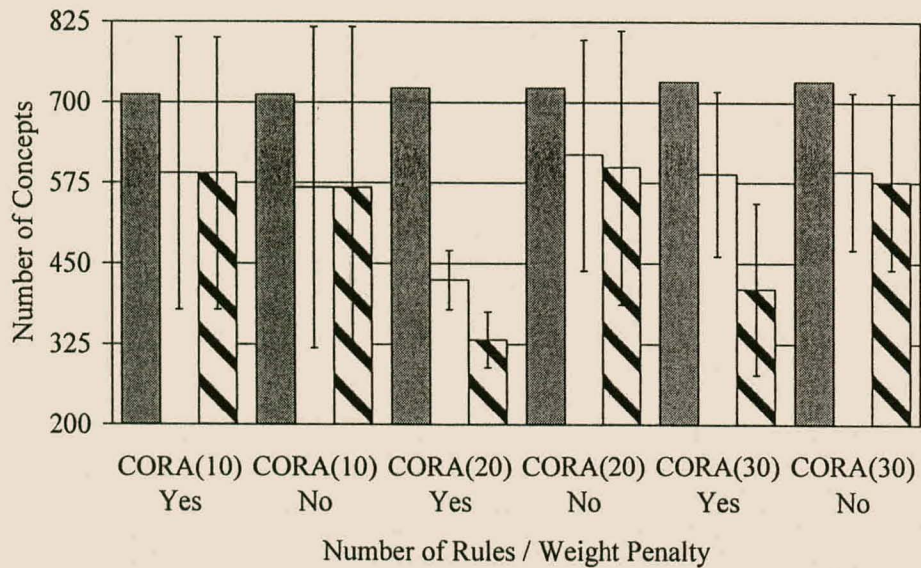




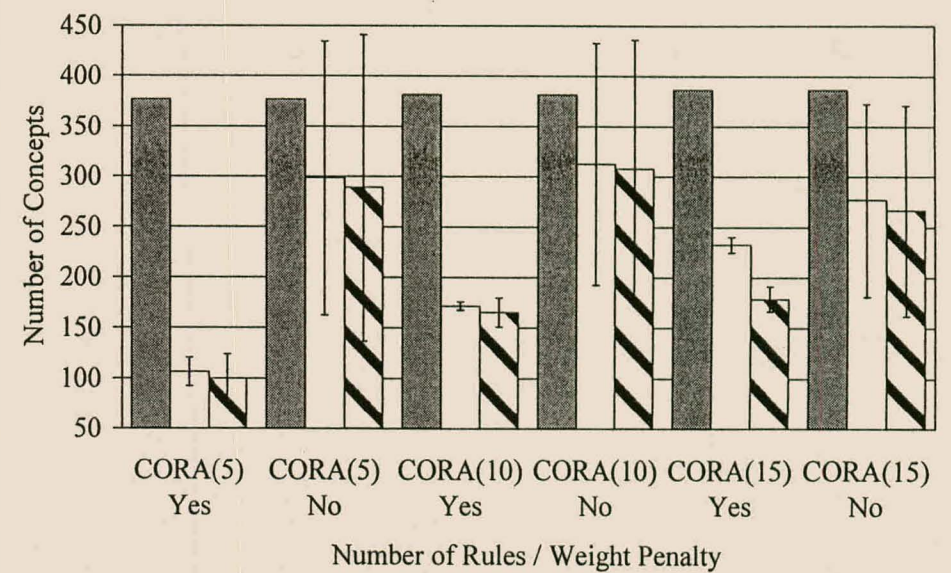
**Figure 6.68 Accuracy vs. Use of Weight Penalty for Housing Data**



**Figure 6.69 Accuracy vs. Use of Weight Penalty for Servo Problem**

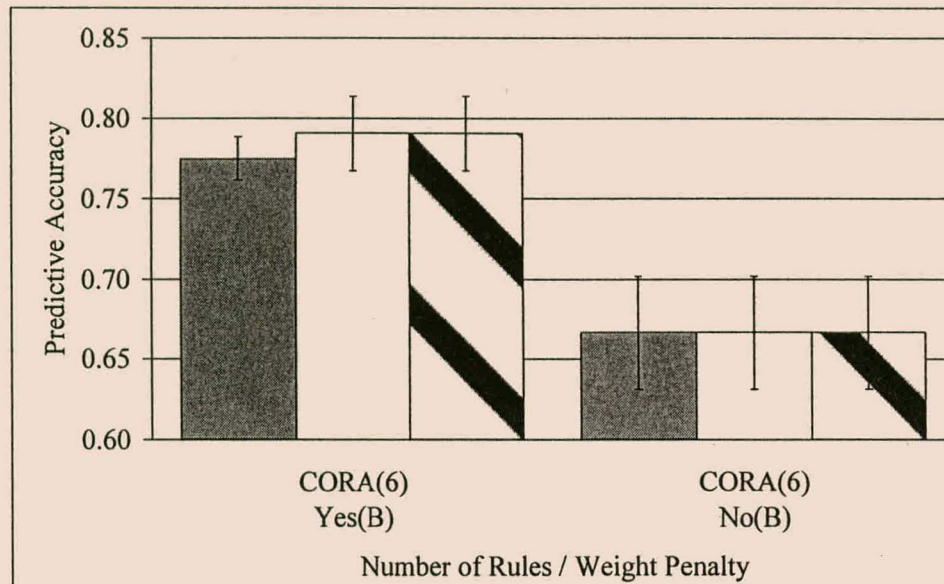


**Figure 6.70 Concepts vs. Use of Weight Penalty for Housing Data**

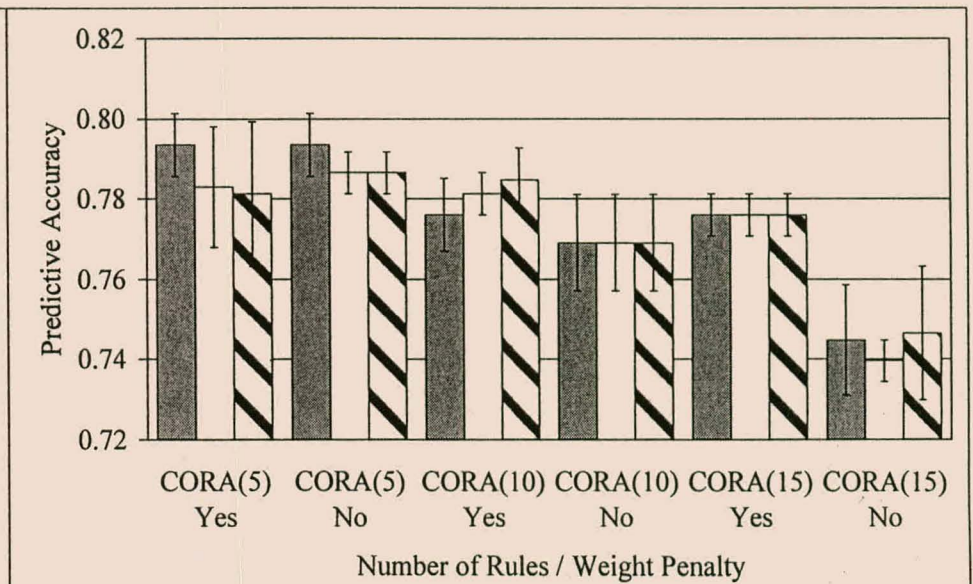


**Figure 6.71 Concepts vs. Use of Weight Penalty for Servo Problem**

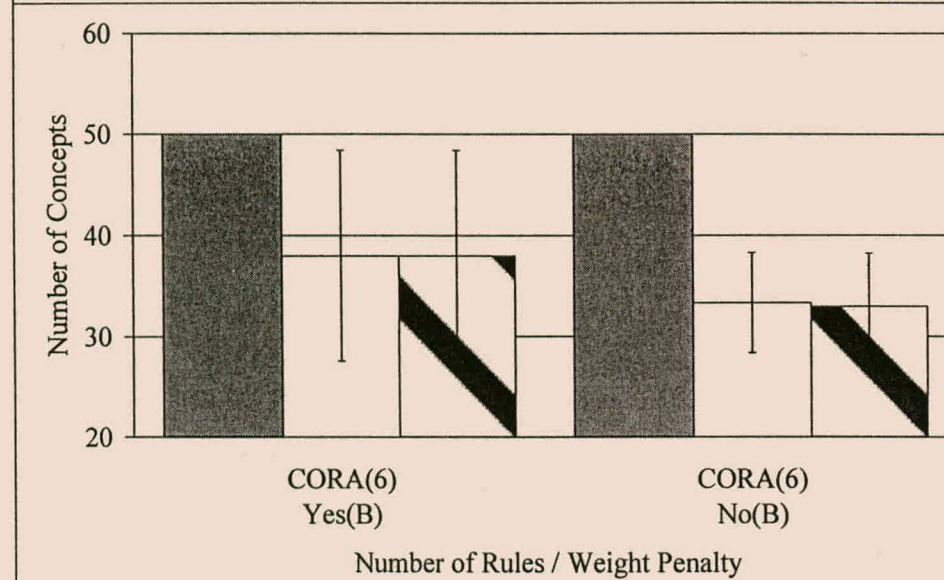




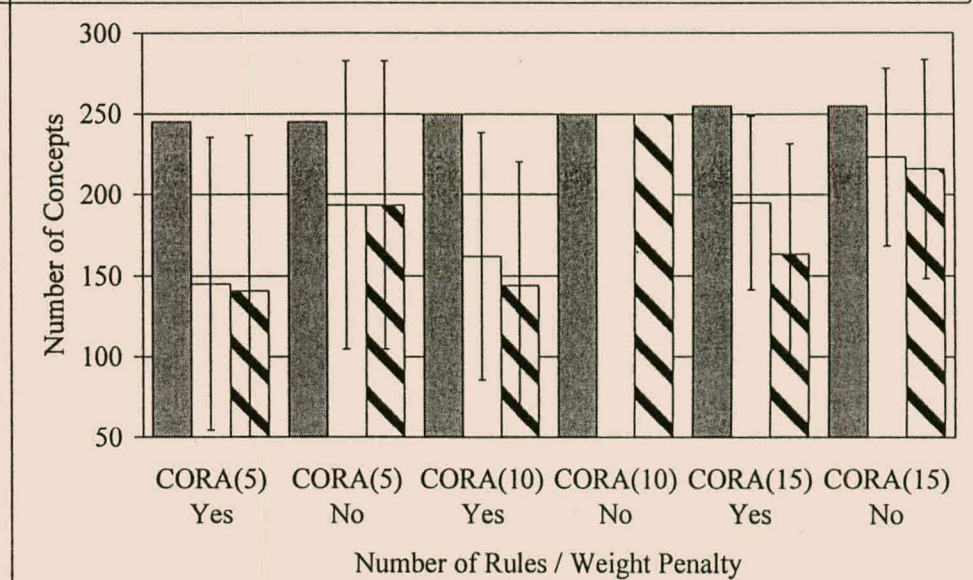
**Figure 6.72 Accuracy vs. Use of Weight Penalty for Slugflow Data**



**Figure 6.73 Accuracy vs. Use of Weight Penalty for Pima Problem**



**Figure 6.74 Concepts vs. Use of Weight Penalty for Slugflow Data**



**Figure 6.75 Concepts vs. Use of Weight Penalty for Pima Problem**

Furthermore, for two or more interacting rules, each with a large consequent, the merging of some of one rule's membership functions can significantly alter the interaction between this rule the other two or more rules. Remember that membership function merging does not explicitly take the interaction between fuzzy rules in the attribute space into account. The negative effects of such fuzzy rule interaction alteration will be more pronounced for rules that have large consequents. This will result in large and most often detrimental variations in model accuracy, in particular after either or both membership function merging and rule reduction have been performed.

- In addition, as expected, consequent magnitude penalisation reduces the variance exhibited by RTS component predictive accuracy for four of the six problems studied, specifically the Abalone (figure 6.64), Auto (figure 6.65), Housing (figure 6.68) and Slugflow (figure 6.72) problems. Furthermore, the reduction in RTS component accuracy generally translates to a reduction in the variance of the accuracy both after membership function merging and rule reduction. This is the case for the Abalone problem (using twenty or thirty rules), the Auto problem (using 22 and 27 rules), the Housing problem, the Servo problem, the Slugflow problem and the Pima problem (using ten or fifteen rules).
- Consequent magnitude penalisation can lead to greater rule model simplification (both membership function merging and rule reduction), in comparison to the experiments where no consequent magnitude penalisation is employed. This statement holds true for the Auto, Housing (using twenty rules), Servo and Pima (using ten rules) problems.

## 6.2.5 Attribute Space Overlap Penalisation

Attribute space overlap penalisation is studied in this section. As described in section 4.3.4.7, the RTS component penalises the fitness of a rule model by a factor of the attribute space overlap of the set of fuzzy rules. The aim of such penalisation is to force the RTS component to build models with at worst the same attribute space overlap as the GNG-derived rule model and in doing so to improve model comprehensibility.

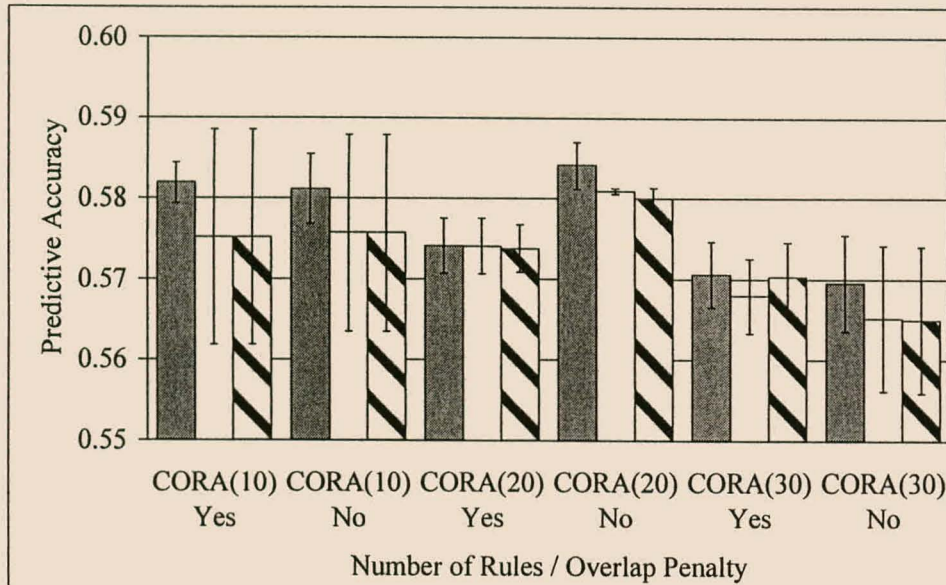
The problems considered in this section are the Abalone, Auto and Pima problems. More problems were studied not studied owing to the time constraints placed on the experiments for this chapter. The experimental results for this section are presented in figures 6.76 through 6.81

as well as figures B.96 through B.98 in appendix B. The model properties that are investigated are predictive accuracy and model complexity. As in section 6.3.2.4, experiments were performed for all three `number_of_rules` settings for each problem. In each figure, the number of assembled rules is indicated in parentheses in the relevant horizontal axis label. All other CORA training parameter settings used in these experiments were identical to those given in table 6.1, with the exception of the attribute space overlap, penalty factor. This parameter was either set to zero or to the value specified in table 6.1.

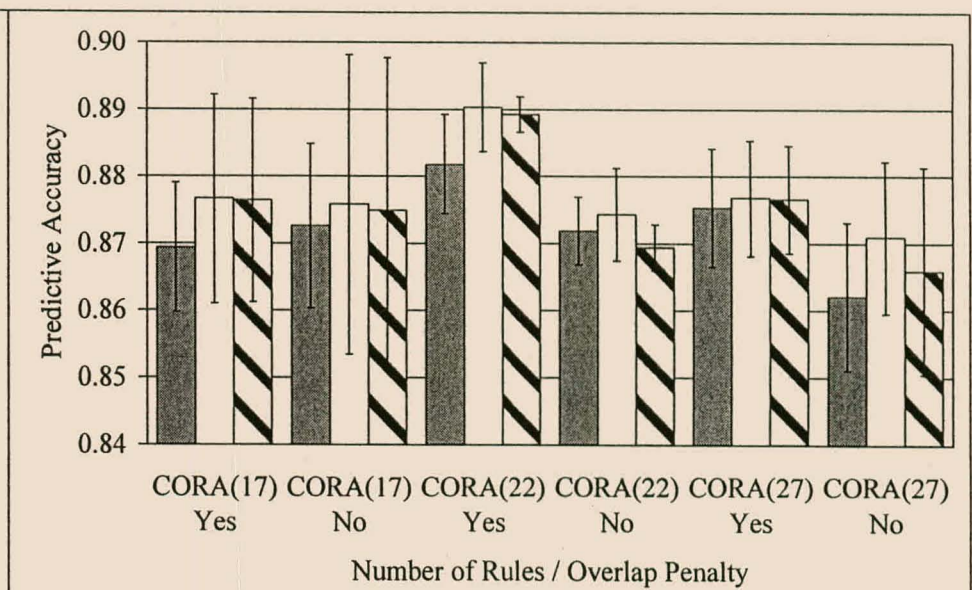
Inspection of figures 6.76, 6.77 and 6.80 reveals that attribute space overlap penalisation, as it is performed by the RTS component of the CORA algorithm, generally has no appreciable effect on predictive accuracy. The two exceptions are the Abalone problem using twenty rules, where penalisation gave significantly worse results, and the Auto problem using 22 rules, where penalisation gave seemingly substantially improved results. Furthermore, attribute space overlap penalisation seems to have had no significant effect on model concept or parameter complexity.

What the results do indicate is that it is generally possible to construct a set of internally disjunctive fuzzy rules from a larger set of purely conjunctive fuzzy rules, using all the membership functions of the latter set, force the former set to have the same attribute space overlap as the latter, decrease model concept and parameter complexity, and still improve on rule model predictive accuracy. This statement is particularly applicable to the Abalone and Auto regression problems. (Refer to figures 6.29 through 6.32, as well as B.1 and B.2 in appendix B for substantiation of the accuracy and complexity claims.)

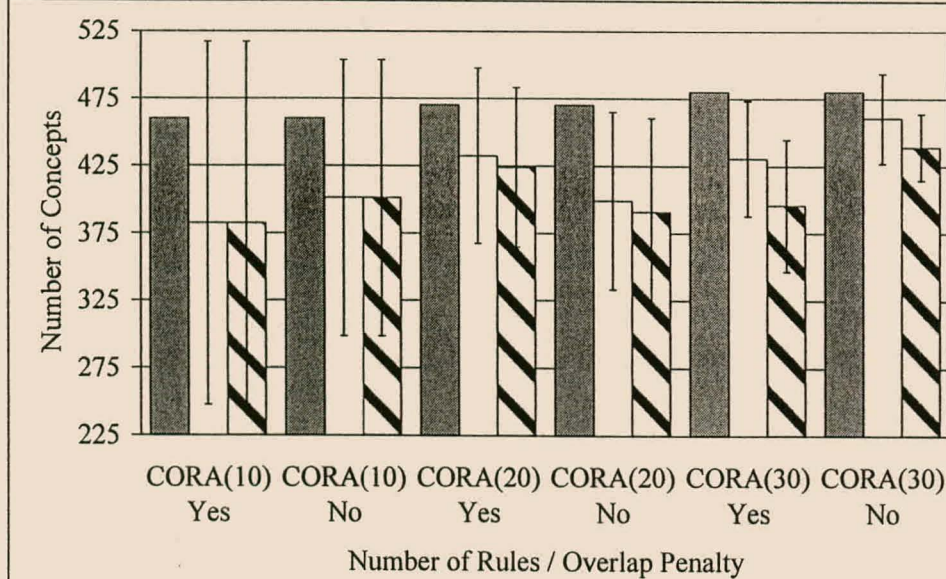




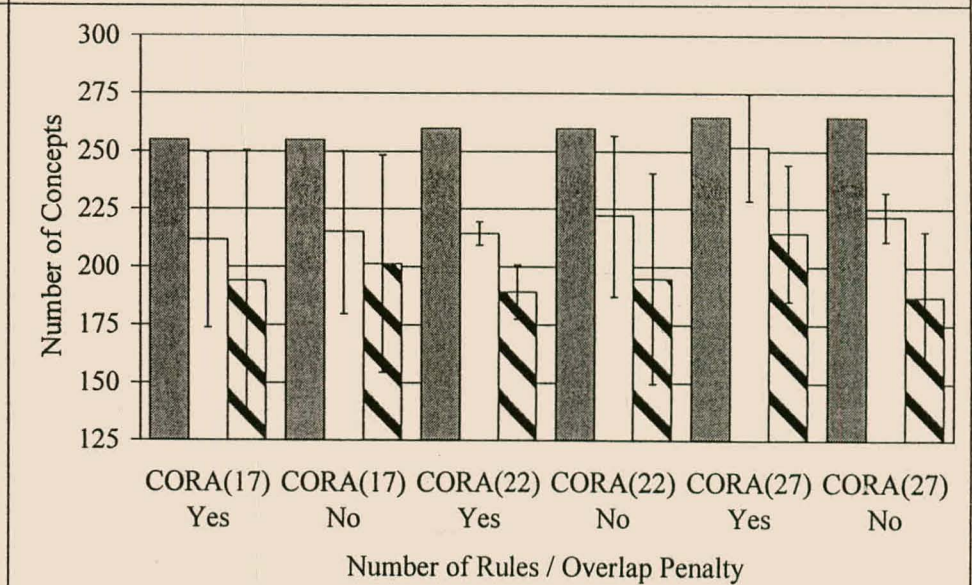
**Figure 6.76 Accuracy vs. Use of Overlap Penalty (Abalone Data)**



**Figure 6.77 Accuracy vs. Use of Overlap Penalty (Auto Data)**



**Figure 6.78 Concepts vs. Use of Overlap Penalty (Abalone Data)**



**Figure 6.79 Concepts vs. Use of Overlap Penalty (Auto Data)**



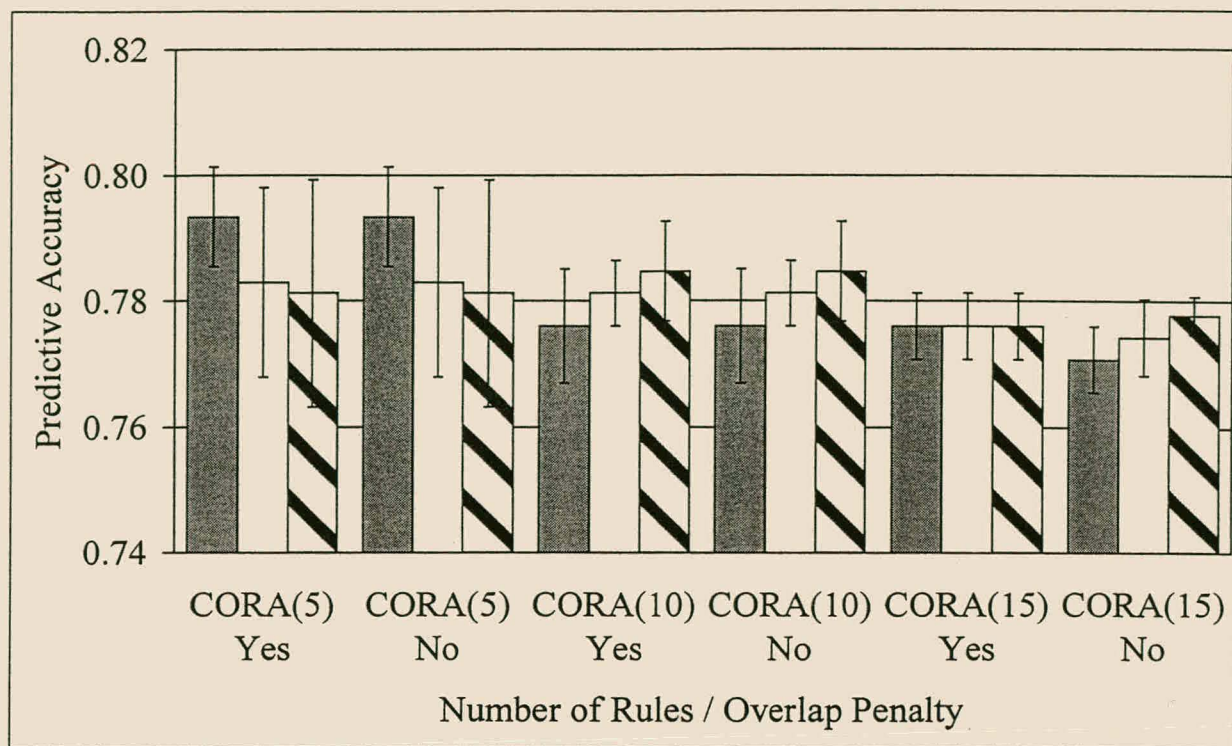


Figure 6.80 Accuracy vs. Attribute Space Overlap for Pima Problem

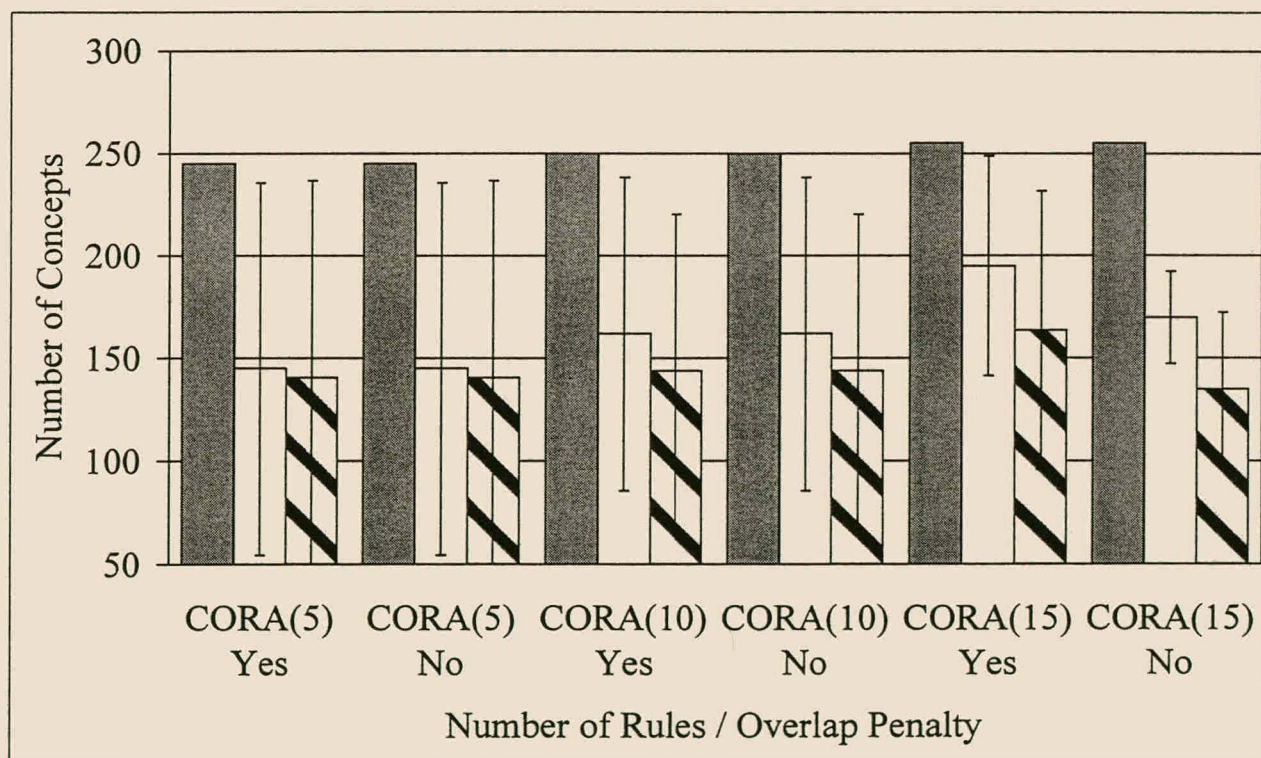
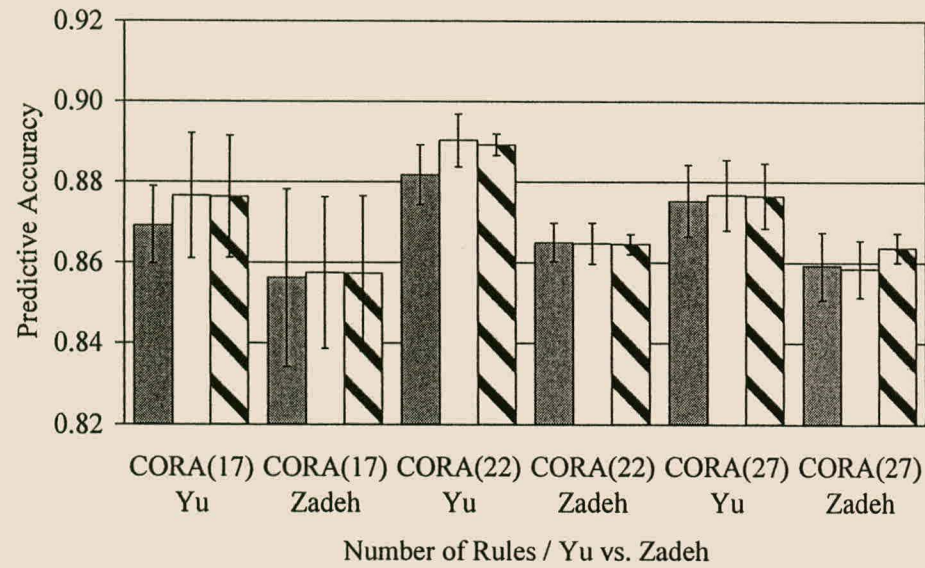


Figure 6.81 Number of Concepts vs. Attribute Space Overlap for Pima Problem

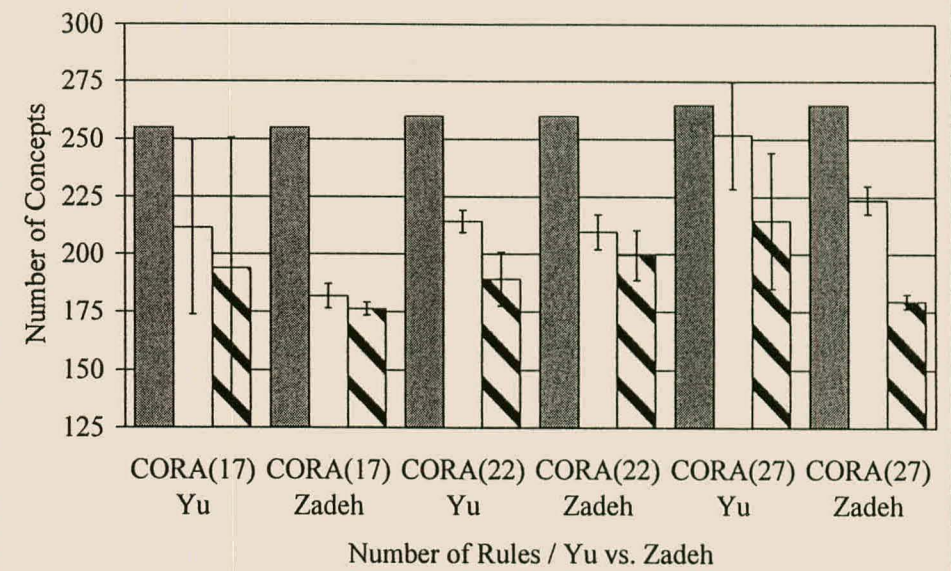
## 6.2.6 Yu Fuzzy Operators vs. Zadeh Fuzzy Operators

As described in section 4.3.1.3, preliminary experimentation brought to light that use of the fuzzy operators proposed by Zadeh (Klir and Yuan, 1995; Kasabov, 1996) produced inferior models in comparison to models based on the fuzzy operators proposed by Yu (1985). This section presents a comparison of the results obtained using these two sets of operators on the Auto problem. The CORA training parameter settings presented in table 6.1 were used in all experiments, with the exception of the number of assembled fuzzy rules and of course the type of fuzzy operator employed. The same three `number_of_rule` settings were used as were studied in section 6.3.1.2, viz. 17, 22 and 27 rules. The experimental results, in terms of predictive accuracy and model complexity, are presented in figures 6.82 through 6.84.

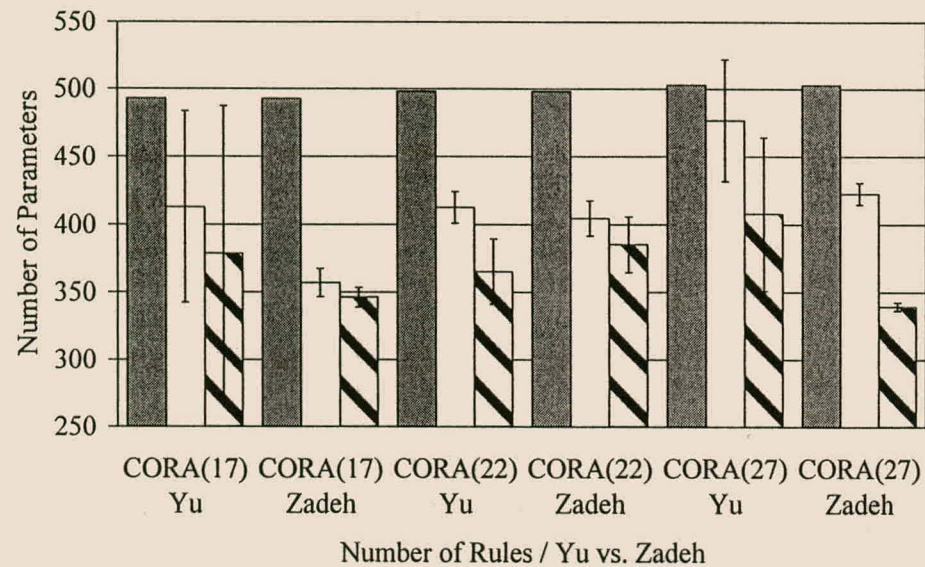
- As shown in figure 6.82, the use of the Yu fuzzy operators improved predictive accuracy results for all three `number_of_rule` settings, although where 17 rules were assembled the high standard deviation of the results makes it difficult to determine the significance of the difference in results. The reason for the difference in results obtained using the Yu operators and those obtained using the Zadeh fuzzy operators is hypothesized to be the following. Consider first rule assembly based on the Zadeh operators. Owing to “maximum / minimum” nature of these operators, the RTS component is able to use very small variations in the membership function profile of a given rule in a given input dimension to change the predicted output of the rule and in turn the interaction between this and other fuzzy rules. In contrast, the Yu OR operator determines the result of an OR operation using both membership functions involved. The determination of the outcome of a Yu AND operation is also distributed between the two relevant operands. This distributed means of calculating fuzzy rule output causes the trained rule model to generalise better than when Zadeh operators are employed.
- Better model simplification (see figures 6.83 and 6.84) was generally obtained using Zadeh operators rather than Yu operators. In particular, for an average 2.1% gain in predictive model concept complexity worsens on average by 6.6% but improves by 2.5% if the parameter complexity measure is used. However, in the author’s opinion the gain in predictive accuracy when Yu operators are used outweighs the increase in model complexity.



**Figure 6.82 Accuracy vs. Use of Yu or Zadeh Fuzzy Operators**



**Figure 6.83 Concepts vs. Use of Yu or Zadeh Fuzzy Operators**



**Figure 6.84 Parameters vs. Use of Yu or Zadeh Fuzzy Operators**



## 6.3 Sundry Issues and a Discussion of Results

This part of chapter 6 investigates miscellaneous issues that have not been covered in the previous sections of this chapter. In addition, a more in depth discussion is given of problems encountered during the experimentation performed for this chapter.

### 6.3.1 Minimisation of Attribute Space Overlap

As described in section 4.3.4.7, the manner in which fuzzy rule attribute space overlapping is performed by the CORA algorithm is to penalise such overlapping if it exceeds the value exhibited by the original GNG-derived rule model. However, the experiments performed for section 6.2.5 showed that for the problems investigated in that section, this form of attribute space overlapping penalisation has no appreciable effect on either model predictive accuracy or complexity. The question can therefore be raised as to what degree of penalisation of attribute space overlapping can be performed without substantially affecting model predictive accuracy. This section studies this question in more detail.

In particular, the attribute space overlap mechanism employed by the RTS component of the CORA algorithm was altered to allow for greater penalisation. Instead of only penalising the CORA model overlap if it exceeds the value exhibited by the GNG rule model, CORA model attribute space overlap is penalised if it exceed a value of one. In other words, the fitness of the CORA rule model is penalised if on average more than one rule needs to be evaluated to determine the rule model's output.

The problems considered in this section are the Abalone, Auto, Housing and Servo problems. More problems were not considered owing to time constraints. For all experiments the CORA training parameters specified in table 6.1 were used, including the overlap penalisation factor. Only one `number_of_rules` setting was tested for each problem. In particular, twenty, twenty-two, twenty and fifteen rules were respectively constructed for the Abalone, Auto, Housing and Servo problems. The results of the experiments are presented in figures 6.85 Through 6.88 as well as figures B.99 and B.100 in appendix B. The model properties that are investigated are the predictive accuracy, model concept complexity and model parameter complexity. The results of two problems are presented in each figure, distinguished by the symbol in parentheses after each horizontal axis label. Ab, Au, H and S respectively stand for the Abalone, Auto, Housing and Servo problems. Furthermore, for each problem the results where the level of overlap is penalised if it exceeds that of the GNG algorithm are presented together with the corresponding

results where the level of overlap is penalised if it exceeds a magnitude of one. As before, each group of three bars represent the results after the RTS, MRG and RED components have successively completed execution. In addition to the figures, table 6.4 presents a comparison of the level of attribute space overlap obtained by the GNG algorithm and each of the two CORA algorithm variants.

Problem	GNG Overlap	CORA Overlap (GNG Level)	CORA Overlap (Level of 1)
Abalone	4.9 (0)	4.4 (0.21)	1.6 (0.14)
Auto	6.6 (0.18)	5.3 (0.092)	1.9 (0.078)
Housing	9.1 (0)	5.9 (0.90)	2.6 (0.19)
Servo	7.3 (0)	4.5 (0.26)	1.4 (0.15)

**Table 6.4 Comparison of GNG and CORA Algorithm Attribute Space Overlap<sup>20</sup>**

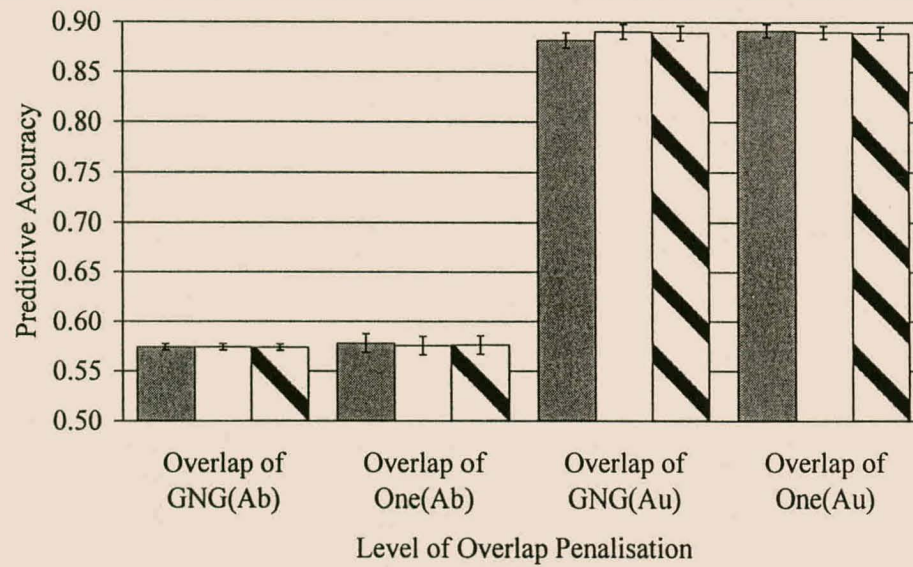
Inspection of the results reveals some interesting phenomena. In particular, for three of the four problems investigated the greater level of attribute space overlap penalisation seems to have no appreciable effect on predictive accuracy. The exception is the Servo problem. The reason for this exception is hypothesised to be the fact that the Servo problem is predominantly discrete. Higher levels of attribute space overlap penalisation seem to detrimentally affect the capability of the RTS component to construct rules that adequately fit this type of data.

---

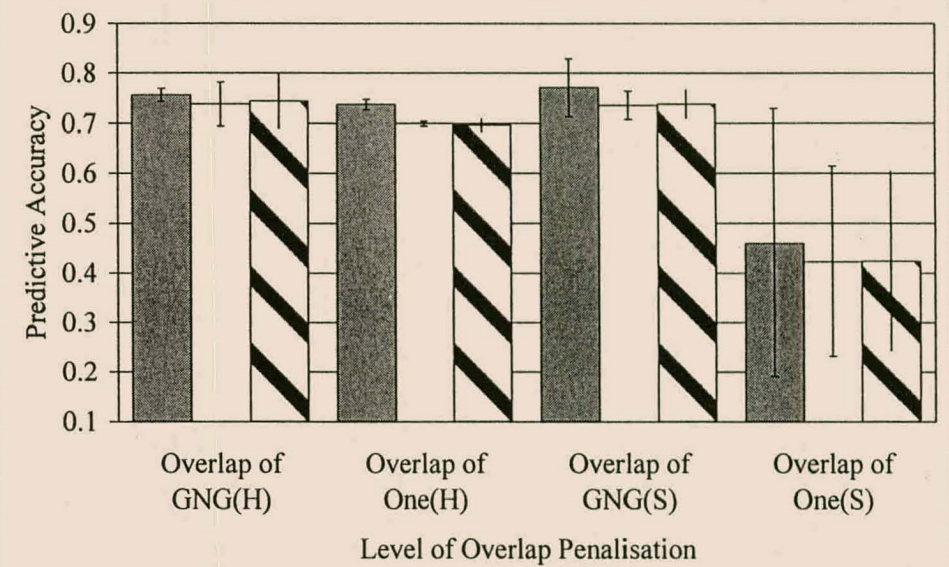
<sup>20</sup> The second column of table 6.4 gives the attribute space overlap of the CORA algorithm variant that penalises overlap if it exceeds that of the GNG algorithm. The third column gives the results of the CORA algorithm variant that penalises overlap if it exceeds a magnitude of one.

The number in brackets given after each overlap value is the standard deviation of the three different random seed runs performed to generate the results.

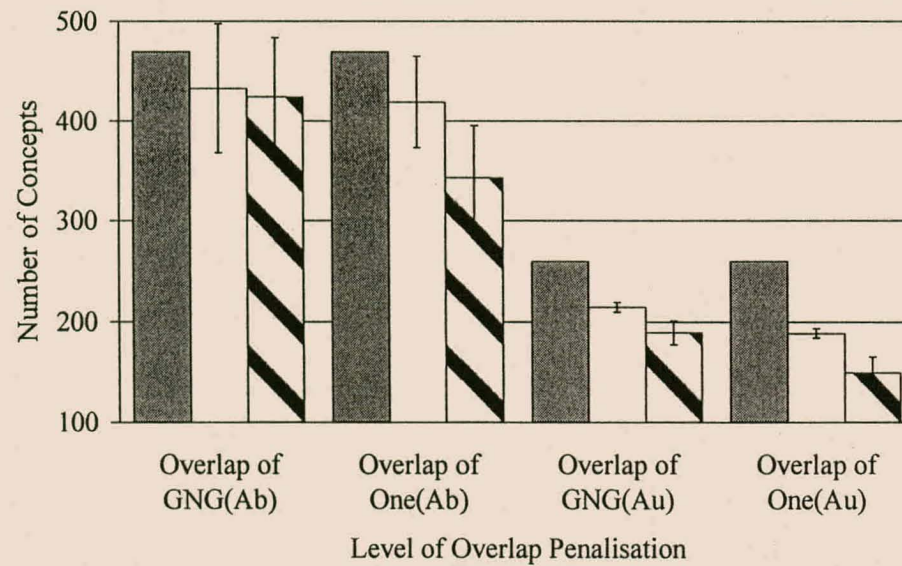




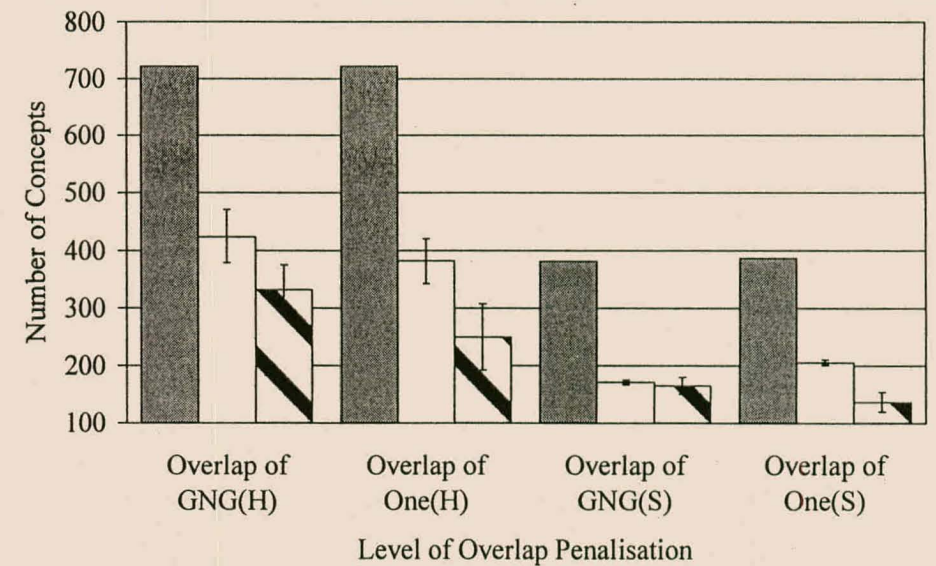
**Figure 6.85 Accuracy vs. Overlap Level (Abalone and Auto Data)**



**Figure 6.86 Accuracy vs. Overlap Level (Housing and Servo Data)**



**Figure 6.87 Concepts vs. Overlap Level (Abalone and Auto Data)**



**Figure 6.88 Concepts vs. Overlap Level (Housing and Servo Data)**

Inspection of the complexity results show that model complexity, both in terms of the number of concepts and the number of parameters, decreases if level of overlap penalisation is increased. This is especially valid for the Auto, Housing and Servo problem results, after all three CORA components have been employed. However, if one examines table 6.4, it is clear that increased overlap penalisation significantly decreases the attribute space overlap exhibited by the CORA rule models.

This means that the CORA algorithm is able to generate rule models with significantly reduced attribute space overlap, less concept and parameter complexity, but improved predictive accuracy, in comparison to the GNG algorithm. However, this statement is only completely valid for the Auto problem. In addition, as mentioned in section 4.3.4.7, preliminary experiments (based on a few small, synthetic problems) performed during the design of the CORA algorithm revealed that in some cases attribute space overlap had a significantly negative effect on model predictive accuracy. More experiments based on different problems should therefore be performed to determine for which kinds of problem the CORA algorithm is able to achieve such results.

As an aside, another means by which attribute space overlap can be reduced, apart from using weighted average defuzzification (Jang and Sun, 1997), is to alter the way in which the GNG algorithm determines the width of the Gaussian of a cell node. The method that is currently used (see section 4.3.4.1 for details) does not explicitly consider attribute space overlap. This means that a GNG rule consisting of a cell node with a wide Gaussian could overlap with, or even completely subsume, a GNG rule that is based on a cell node with a relatively narrow Gaussian. This is especially the case if the latter cell node is close to the former, but based on the GNG adjacency structure is not seen to as a neighbour of the former cell node.

More careful analysis of the overlap that is caused by the updating of a GNG cell node Gaussian's width would allow one to determine in which regions of the GNG network a high level of attribute space overlapping is prevalent. This information could then be presented to the GNG training algorithm as an additional resource<sup>21</sup> term with the aim of biasing GNG network construction towards models exhibiting less attribute space overlap. Lower GNG rule model overlap would make it easier for the CORA algorithm to generate models exhibiting lower attribute space overlap.

---

<sup>21</sup> See section 4.1.1 for a definition of "resource".

## 6.3.2 RTS Component Training Characteristics

Two aspects of the training speed of the RTS component of the CORA algorithm will be discussed here. The first is whether the use of the adjacency matrix generated by the GNG algorithm (section 4.3.1.2) significantly reduces the combinatorial complexity of the search performed by the RTS component. The second aspect that is discussed is whether nongreedy search, as opposed to greedy search, is required to find the optimal solution.

### 6.3.2.1 Combinatorial Complexity Reduction Using the GNG Adjacency Matrix

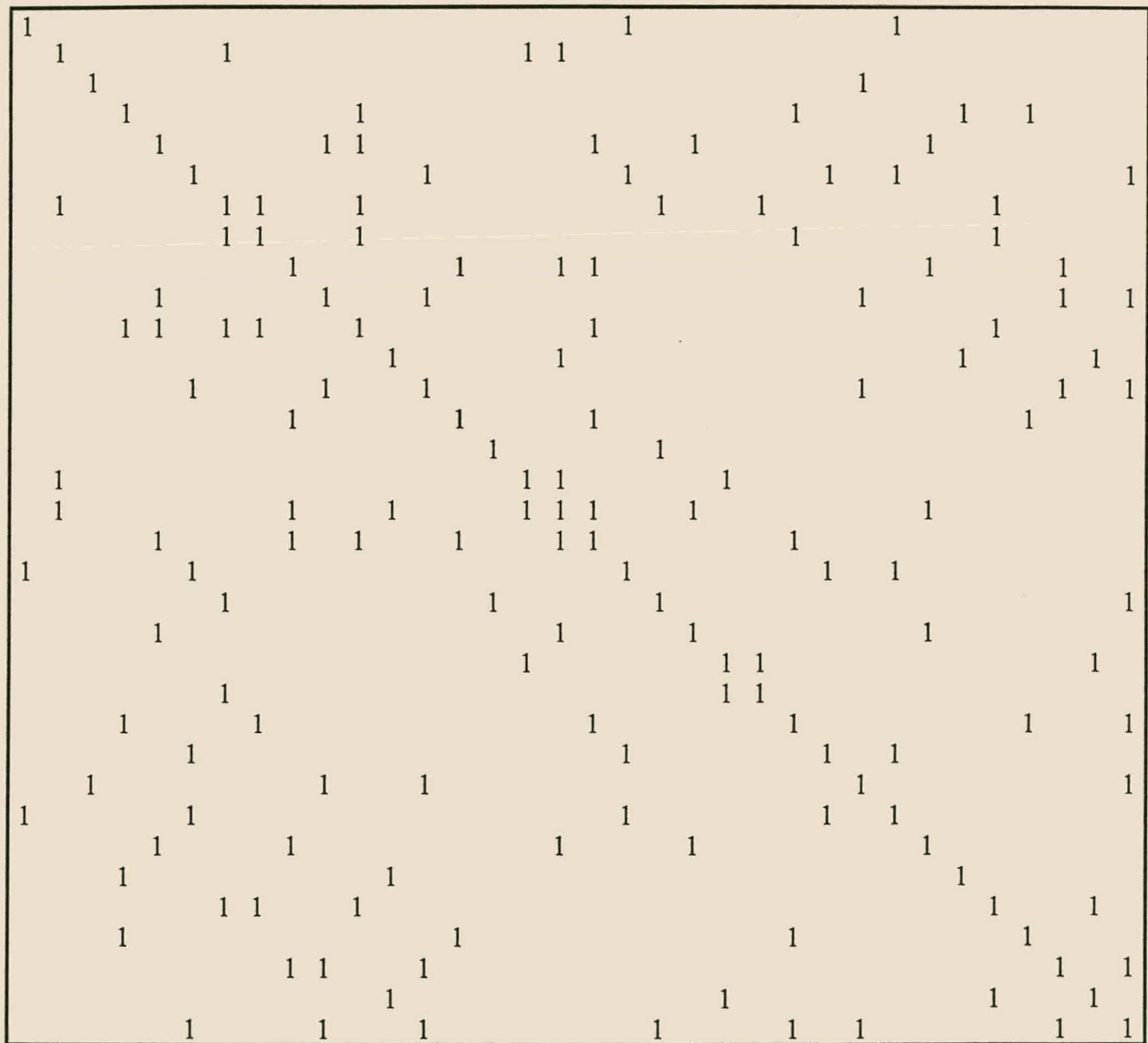


Figure 6.89 GNG-generated Adjacency Matrix for Auto Problem Using Thirty Rules



Figure 6.89 gives a typical adjacency matrix<sup>22</sup> generated by the GNG algorithm. A matrix element indicates whether the cell nodes represented by the row-column combination are topologically adjacent. (A cell node is assumed adjacent to itself to allow the CORA algorithm's membership function swapping permissibility function to operate correctly.) Specifically, a matrix element value of one indicates that two GNG cell nodes are adjacent whereas a blank value indicates that they are not.

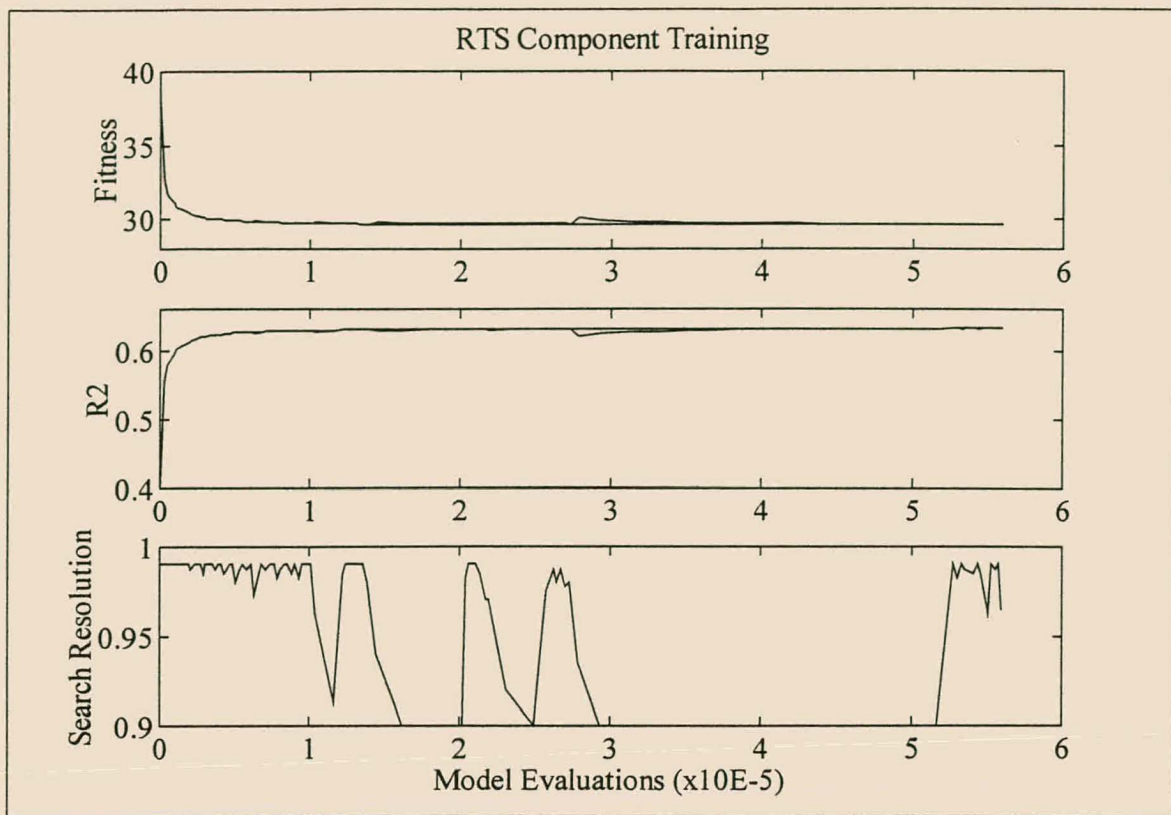
It should be clear from figure 6.89 that significantly more cell nodes are not adjacent to each other than cell nodes that are adjacent. This means that the combinatorial complexity of evaluating each rule model that can be generated using any one of all permissible swaps (or moves) is significantly less than the worst case scenario described in section 4.3.4.8 (section 4.3.4.9 for moves). This means that use of the GNG adjacency matrix significantly reduces the computational cost of the combinatorial search performed by the RTS component of the CORA algorithm.

### 6.3.2.2 Training Profiles Exhibited by the RTS Component of the CORA Algorithm

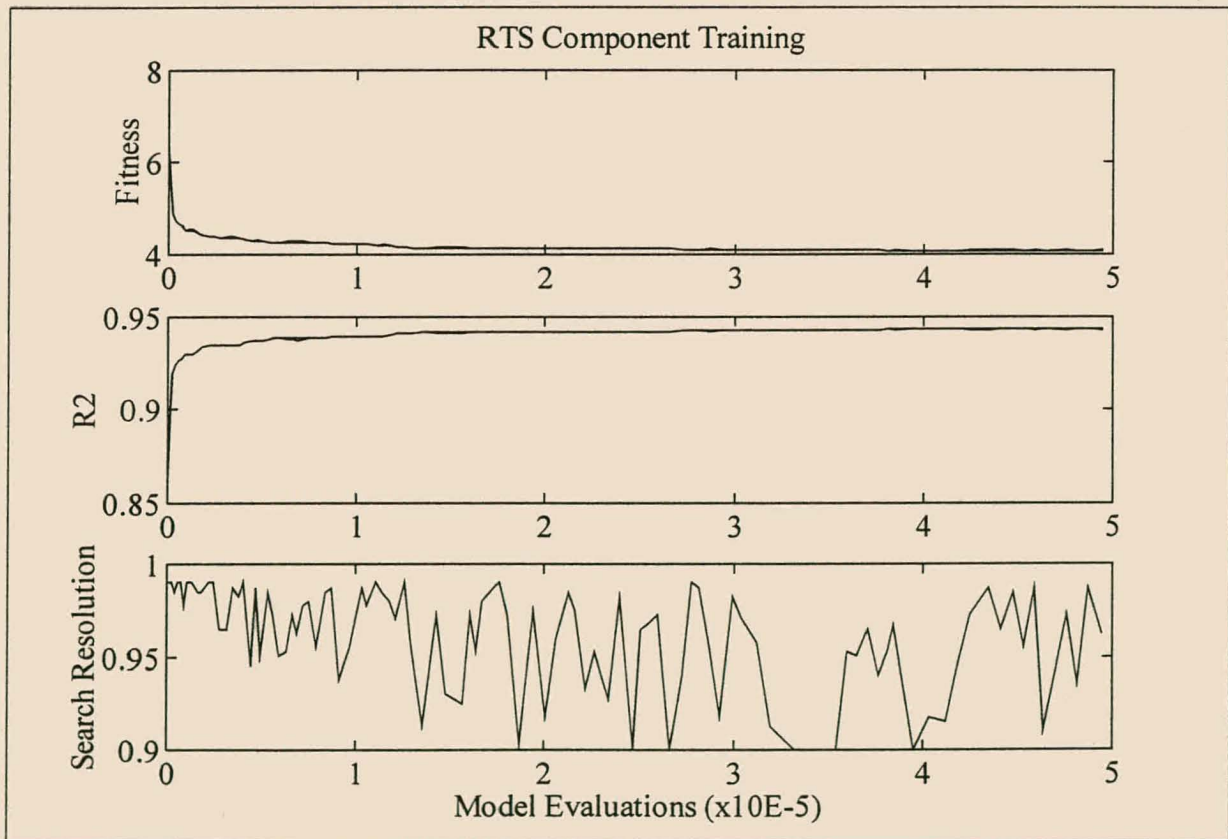
Typical model fitness, regression accuracy ( $R^2$ ) and search resolution profiles are given in figures 6.90 through 6.93. The results are for the Abalone, Auto, Housing and Servo problems using 20, 22, 20 and 15 rules, respectively. The training parameter settings used for these problems are identical to those given in table 6.1. Each figure give results for a single run of the CORA algorithm based on a random seed of nine. The horizontal axis of each subplot represents the number of model evaluations completed by the RTS component of the CORA algorithm. Note that the RTS component attempts to minimise model fitness, but in doing so tries to maximise model regression accuracy.

---

<sup>22</sup> The adjacency matrix was generated for the Auto problem using thirty radial basis functions. The training parameter settings given in table 6.1 were used.

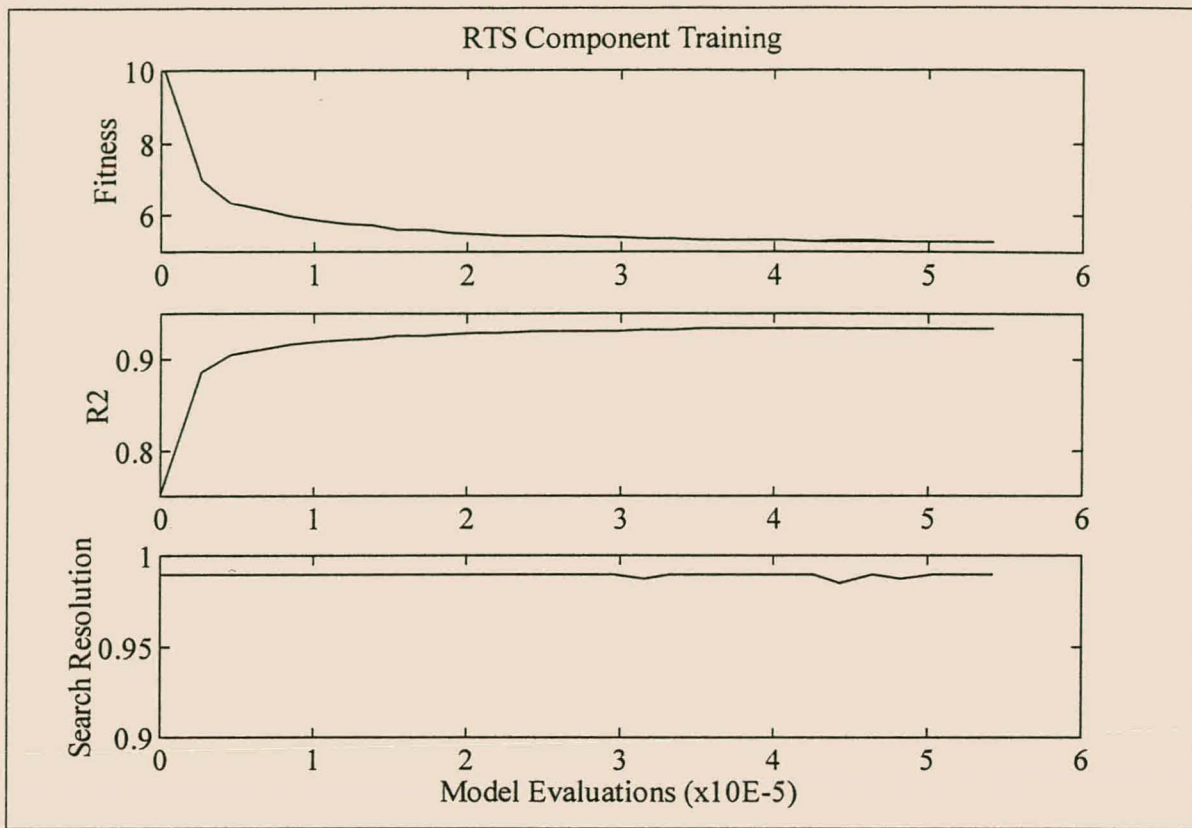


**Figure 6.90 Fitness, Training R<sup>2</sup> and Search Resolution for the Abalone Problem**

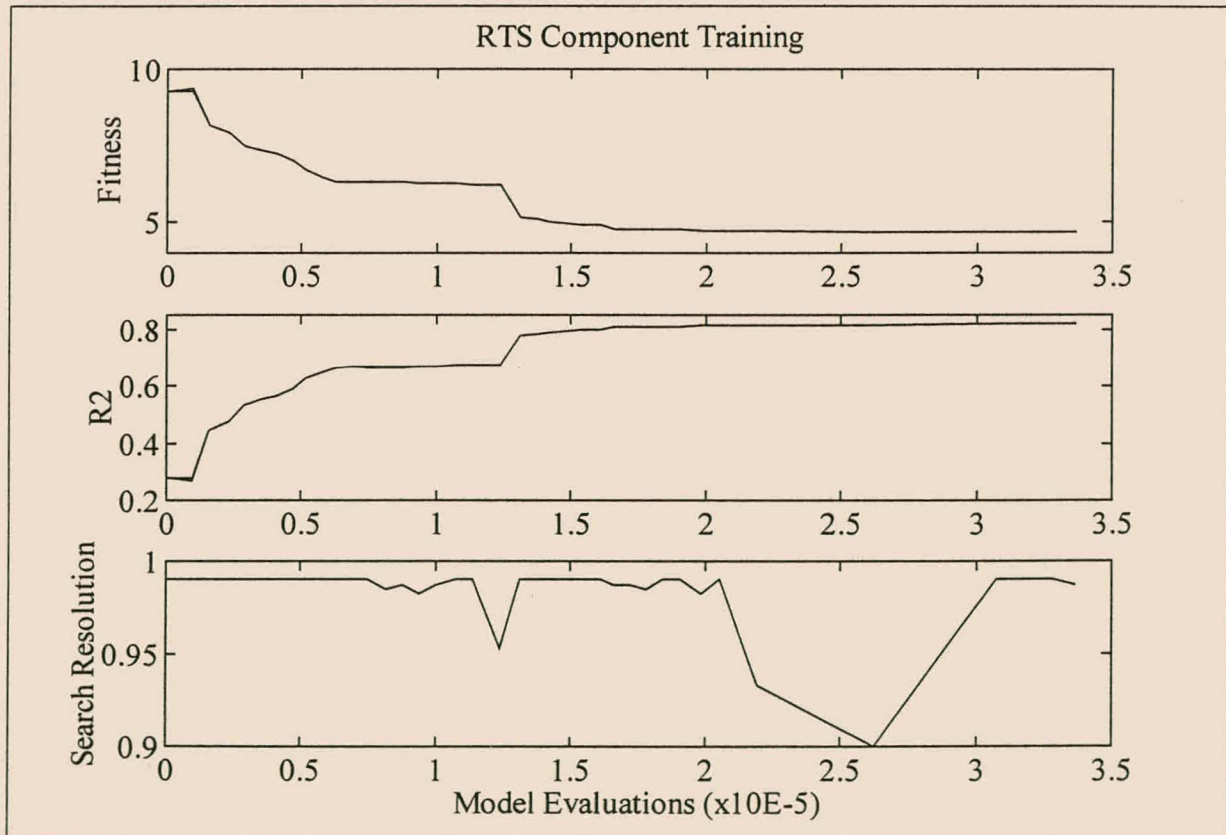


**Figure 6.91 Fitness, Training R<sup>2</sup> and Search Resolution for the Auto Problem**





**Figure 6.92** Fitness, Training  $R^2$  and Search Resolution for the Housing Problem



**Figure 6.93** Fitness, Training  $R^2$  and Search Resolution for the Servo Problem

Inspection of the bottom subplots of each of above figures shows that a number of locally optimal solutions are encountered en route to the best solution found by the RTS combinatorial search. Remember that the search resolution is reset to its maximum value of 0.99 if and only if a solution is found that is more fit than any solution found by the RTS algorithm in previous search iterations. This means that simple greedy search will get stuck in local optima and not find as optimal a solution as nongreedy search will be capable of finding. This is a partial justification for the use of the nongreedy RTS combinatorial search algorithm over greedy search algorithms.

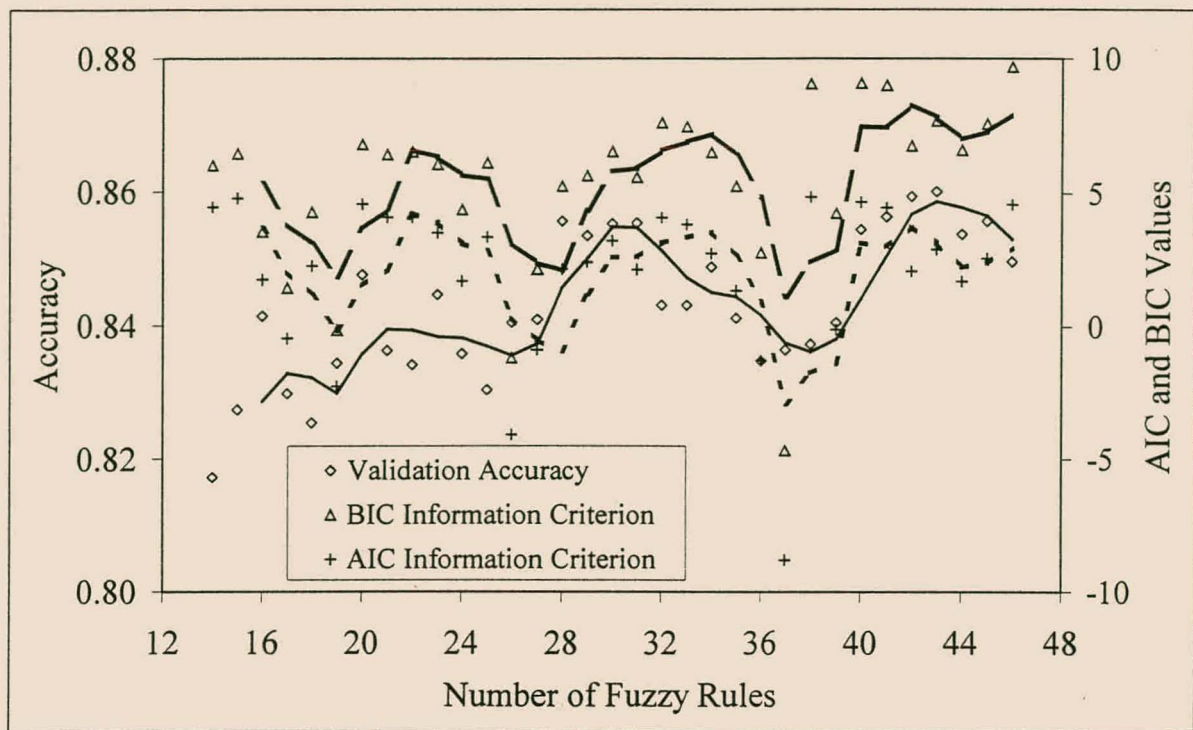
### 6.3.3 Selecting the Optimum GNG Radial Basis Function Network Model and Membership Function Merging

This section discusses the criteria used to determine the optimum GNG radial basis function network size as well as optimal level of merging and rule reduction by the CORA algorithm.

Consider first the optimality of the AIC and BIC information criteria (Akaike, 1973, 1974, 1977). As described in section 4.3.4.4 both the AIC and BIC criteria are employed to determine the optimal GNG radial basis function network size and the threshold at which the CORA algorithm performs membership function merging. Even though satisfactory results were obtained using these criteria, it should be remembered that these criteria were originally proposed for linear systems only. It should be clear that both the GNG and CORA rule models contain nonlinear components. In order to determine the effectiveness of the two criteria, a number of different size GNG radial basis function networks were trained using the data from the Auto problem. The default GNG training parameter settings given in table 6.1 were used for these experiments, with the exception of the number of training epochs. The number of training epochs was chosen proportional to the size of the GNG network under consideration. Larger networks were trained for more epochs. The results of these experiments are presented in figure 6.94<sup>23</sup>.

---

<sup>23</sup> Figure 6.94 compares the validation accuracy obtained by the GNG rule model with the calculated AIC and BIC values for a number of different size GNG networks. The bottom solid line is a moving average line with period three of the validation accuracy results. The top dashed line is a moving average line with period three of the BIC information criterion values. The remaining dotted line is a moving average line with period three of the AIC information criterion results.



**Figure 6.94 Validation Accuracy and Information Criteria Results vs. Number of GNG Fuzzy Rules**

Inspection of figure 6.94 shows that both the AIC and BIC criteria exhibit at least four local minima. In addition, the optimal solution predicted by these two criteria is a GNG network consisting of 37 nodes. Figure 6.94 shows that this GNG network did not generate the best validation accuracy results. On the contrary, the most accurate GNG network consists of 43 cell nodes and has a validation  $R^2$  of 0.86, as opposed to the validation  $R^2$  of 0.84 of the 37-node GNG network. The phenomenon of multiple local optima means that exclusive use of the AIC and BIC criteria, as a means of finding the “right-sized” GNG model, could cause suboptimal GNG models to be selected. However, satisfactory results were obtained in the GNG network construction experiments performed for this chapter, owing to the fact that the use of these two information criteria is coupled with a third criterion. As described in section 4.3.4.4, the third criterion is the predictive performance of the GNG rule model on unseen data.

On the other hand, the AIC and BIC information criteria are currently the only way that the level of membership function merging by the MRG component of the CORA algorithm is determined. It is hypothesised that the local optima generated by the AIC and BIC criteria contributed to the MRG component in some experiments producing results that exhibit high standard deviations (see for example figures 6.30 and 6.32 in section 6.2.1). Unfortunately, owing to time constraints, experiments to evaluate this aspect of membership function merging could not be performed.

### 6.3.4 Variance of Results Generated by the CORA Algorithm

The final issue that is discussed in this chapter is the high variance or standard deviation of the results generated by the CORA algorithm for some problems. Three factors are thought to be the primary cause of the high variance of the results generated by the CORA algorithm.

- Both the GNG algorithm and the RTS component of the CORA algorithm generate results with less variance for problems that have a relatively high data exemplar density<sup>24</sup>. In particular, the results with the lowest variance were generally obtained for experiments based on the Abalone problem. (The Abalone problem has a data density of 261.) Consider the results presented in figure 6.95. Figure 6.95 gives the variance of the predictive accuracy of the models generated by the three CORA algorithmic components vs. the data density of the problems that were considered. The results for the Autocatalytic problem, first considered in chapter 7, are also shown. (The Autocatalytic problem has a data density of 159.)

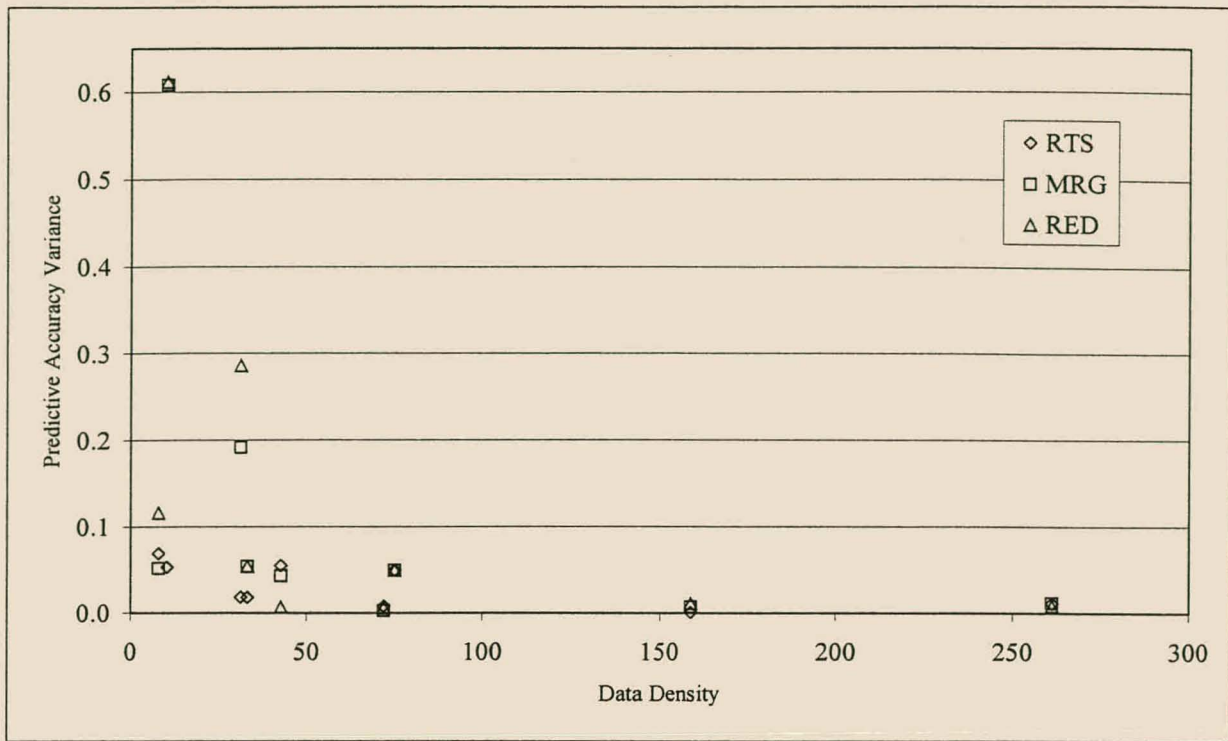
Inspection of figure 6.95 shows that that the current version of the CORA algorithm is more suited to problems for which there exist a large number of data. If there are few data the CORA algorithm struggles to find the true underlying model that is represented by the data. This phenomenon holds true for any of the three algorithmic components. (The trend of decreased accuracy variance as the data density increases is the most pronounced for the MRG and RED components.)

Similar conclusions were reached during the analysis of the results obtained by existing rule construction techniques (section 3.3.3). In that study it was found that less-than-greedy rule construction techniques struggle to build models that perform significantly better than models built using simple greedy techniques for problems that exhibit a low data exemplar density.

---

<sup>24</sup> Data exemplar density is defined as the quotient of the number of training exemplars divided by the number of problem attributes.





**Figure 6.95 Variance of Predictive Accuracy Results vs. Data Set Density**

- As mentioned in section 6.3.3, the criteria used to determine the level of membership function merging performed by the MRG component of the CORA algorithm generate results that exhibit local optima. Exclusive use of these criteria can cause membership function merging to be performed at various thresholds. This can cause models to be generated that are significantly different and therefore exhibit varying levels of predictive performance and model complexity.
- The combinatorial complexity of the rule assembly problem solved by the RTS component of the CORA algorithm is large. For example, consider the experiment where twenty fuzzy rules are assembled for the Abalone problem. (Results of this experiment are presented in section 6.3.2.2, specifically in figure 6.90.) On average the RTS component evaluated 281 models per iteration. Assume that on average these 281 models formed only 10% of all possible models<sup>25</sup> that could be created from the current rule model. If this is the case then the RTS algorithm randomly selected these models from a total of 2810 rule models. This random selection of models occurred during each of 2000 RTS iterations. It should be clear that it is

<sup>25</sup> The minimum swap (or move) threshold for this particular experiment was 0.9. At worst the RTS algorithm would therefore have to evaluate 10% of all possible solutions that could be created from the current rule model.

difficult for the RTS algorithm to consistently construct the same fuzzy rule model if different random seeds are used<sup>26</sup>.

## 6.4 Summary and Conclusions

This chapter compared the performance of the CORA algorithm with other rule construction techniques, as well as techniques that do not explicitly build models in the form of a set of “if...then...” rules. In addition, the effect of the CORA algorithm training parameters and the usefulness of the different algorithmic components were evaluated and discussed. The following summary of results and conclusions can be drawn from the experiments and discussion presented in this chapter:

- For the Spiral problem the CORA algorithm is capable of retaining the geometrical form (two intertwined spirals) and training accuracy of the GNG rule model. In addition, the CORA model contained 30 rules in comparison to the 105 rules of the GNG model.
- The CORA algorithm is capable of constructing rule models with comparable or nominally better predictive accuracy than all the other techniques considered, with the exception of the two multilayer perceptron variants. However, the CORA algorithm performs poorly in comparison to the other techniques that were evaluated on problems, the Servo problem in particular, that are predominantly discrete and contain few data.

For eight of the nine problems investigated, the CORA algorithm was able to assemble a set of fuzzy rules that is more accurate than the GNG rule model from which the membership functions of the CORA rule model were drawn. CORA rule models are always less complex than their GNG counterparts in terms of number of fuzzy rules, concept complexity and parameter complexity.

For the nine regression and classification problems considered in this chapter, the models derived by the CORA algorithm are more complex than their CART and

---

<sup>26</sup> A second example of the combinatorial complexity of the rule assembly exercise is based on an experiment performed on the Auto data, the results of which are presented in figure 6.91. For this experiment the RTS component evaluated on average 195 models per RTS iteration. A total of 1072271 models were evaluated over 5000 RTS iterations.

BEXA derived counterparts in terms of concept complexity. However, in some cases the CORA models contain fewer rules, but exhibit comparable or better predictive accuracy. This means that for some problems the rules derived by the CORA algorithm have greater modelling capability than the rules used by CART or BEXA.

In terms of parameter complexity, the CORA rule models are generally more complex than the models derived by the other techniques (not the GNG algorithm) although there are a few exceptions (see section 6.1.3 for details).

- The CORA algorithm is able to generate rule models that exhibit less attribute space overlap than their GNG-derived counterparts. However, for some problems (the Housing, Servo, Ionosphere, Slugflow and Pima problems) and for certain CORA training parameter settings, the reduction in attribute space overlap is detrimental to model predictive performance. Such models exhibit less overlap but also have poorer predictive accuracy than their GNG-derived counterparts.

Further experimentation (section 6.3.1) brought to light that for the Abalone, Auto and Housing problems and certain CORA training parameter settings, the magnitude of attribute space overlap is reduced by more than two-thirds, in comparison to the overlap exhibited by the GNG rule models, but without appreciably affecting model predictive accuracy. In these overlap reduction experiments the attribute space overlap was reduced to between 1.6 and 2.6. This means that it is possible for the CORA algorithm to generate models with attribute space overlap that approaches that of the CART and BEXA rule models. (The CART and BEXA rule models always have an overlap of one.) This can be done even though the GNG model (upon which the CORA rule models are based) overlap ranges between 4.9 and 9.1 for the Abalone, Auto and Housing problems.

- For the Abalone, Auto, Housing and Servo regression problems the correct choice of the number of assembled rules allows the CORA algorithm to construct models with nominally superior predictive accuracy in comparison to those generated by the GNG algorithm. Furthermore, in general an increase in the number of assembled fuzzy rules leads to an increase in predictive accuracy. However, the use of too many rules leads to overfitting on the training data.
- In the “number of fuzzy rules” experiments performed for section 6.2.1, membership function merging generally has a negative effect on predictive performance.

However, for the Auto and Ionosphere problems membership function merging decreases model complexity but increases predictive accuracy. Furthermore, for the Abalone, Housing and Pima problems merging on average and at worst decreases predictive accuracy by 1.6% for a corresponding 31% decrease in model concept complexity. In the author's opinion the drop in predictive accuracy is therefore in most cases outweighed by the increase in model simplicity.

For the experiments performed in section 6.2.1, rule reduction increased with increasing number of assembled rules. Furthermore, rule reduction, in contrast to membership function merging, does not always decrease model predictive accuracy. At worst (for the Ionosphere classification problem using thirty rules) rule reduction decreased predictive accuracy by 0.87%. However, rule reduction is generally able to appreciably decrease model complexity. For the problems considered in section 6.2.1, model complexity decreased by on average 19% in terms of model concepts and by 22% in terms of model parameters.

- An increase in the swap (or move) threshold employed by the CORA algorithm does not consistently lead to significantly different (more or less accurate, or less complex) rule models. This means that a relatively stringent swap (or move) threshold value such as 0.9 can be used to substantially decrease computational complexity without seemingly altering the properties of the models that are derived. However, it must be kept in mind that all RTS combinatorial searches currently start with a search resolution of one percent. It seems that the adaptive modification of the search resolution by the RTS component, from an initial value of one percent up to the swap (or move) threshold, plays a more significant role in determining final rule model properties than what the minimum swap (or move) threshold does. This statement is substantiated by the lack of any consistent trends in the results of the experiments performed to study the influence of the swap (or move) threshold (see section 6.2.2).
- The use of moves or both swaps and moves by the RTS component as a means of assembling fuzzy rules, instead of only using swaps, does not seem to have a consistently beneficial or detrimental effect on model predictive accuracy. In the experiments performed for section 6.2.3, the use of moves instead of swaps generally leads to greater model complexity reduction by the MRG and RED components of the CORA algorithm.



- The use of consequent magnitude penalisation generally improves rule model predictive accuracy, especially if many fuzzy rules are assembled. In addition, it was found that consequent magnitude penalisation reduced the variance of the RTS component predictive accuracy results for four of the six problems studied in section 6.2.4. This generally leads to reduced variation in the predictive accuracy results of the MRG and RED components. Furthermore, in some cases the use of consequent magnitude penalisation leads to greater rule model simplification.
- On the Auto problem the use of the fuzzy operators proposed by Yu rather than the operators proposed by Zadeh allows the CORA algorithm to construct more accurate rule models. However, greater model simplification is possible if the latter set of fuzzy operators is employed. In the author's opinion, the average gain in predictive accuracy (2.1%) outweighs the average increase in model concept complexity (6.6%) if Yu rather than Zadeh operators are used. Interestingly, if Yu operators are used the model parameter complexity on average improves (i.e. decreases) by 2.5%.
- The use of the GNG adjacency matrix allows the CORA algorithm to reduce significantly the complexity of the combinatorial search performed by the RTS component. The combinatorial complexity of evaluating each rule model that can be generated using any one of all permissible swaps (or moves) is significantly less than the worst case scenario described in section 4.3.4.8 (section 4.3.4.9 for moves).
- Experiments on the Abalone, Auto, Housing and Servo problems showed that for all of these problems the RTS component encountered a number of local optima during its search for the rule model with the best fitness. This means that nongreedy search will obtain better solutions than simple greedy search will for these problems.
- Both the AIC and BIC information criteria generate results that exhibit a number of local optima if they are used to determine the "right-sized" GNG model or to choose the optimal level of membership function merging by the MRG component. This could lead to suboptimal GNG rule models as well as higher than expected variation in the results obtained by the MRG component of the CORA algorithm.

Other criteria for determining the "right-sized" model were studied during the development of the CORA algorithm. However, it was found at that stage that the AIC and BIC criteria were easy (in the author's opinion) to implement and produced (for those experiments) satisfactory results. In the light of the conclusions reached in

this chapter regarding the suboptimality of the AIC and BIC information criteria it is suggested that more advanced criteria, such as the minimum description length (Rissanen, 1983; Quinlan and Rivest, 1989); Judd and Mees, 1995) or minimum message length (Oliver and Hand, 1994) criteria, be evaluated in future development of the CORA algorithm.

- In many experiments the models generated by the CORA algorithm using different random seeds only, produced results that exhibited greater than expected variation. It is thought that there are three reasons for this phenomenon. First, the CORA algorithm struggles to model problems that have a low data exemplar to number of attributes ratio, leading to greater than expected variation in results. In section 6.3.4 it is shown that this problem is not restricted to the CORA algorithm. The same phenomenon occurs using other advanced rule construction techniques. Second, the exclusive use of the AIC and BIC information criteria by the MRG component of the CORA algorithm can result in membership function merging being based on different local optima found in the AIC and BIC results. If this is the case, membership function merging will occur at varying thresholds, leading to increased variation in results. Third, the combinatorial complexity of constructing rules in the way that the CORA algorithm does is large. This means that it will be difficult for the CORA algorithm to consistently find the same solution if different random seeds are used.

# Chapter 7

## Modelling of a Chaotic Reaction System

This chapter studies the application of the CORA algorithm to the modelling of a chaotic reaction system. (Consult (Abarbanel, 1996) for more information regarding chaotic systems.) The reaction system that is investigated is an autocatalytic process that takes place in a continuously stirred tank reactor (CSTR). The first part of the chapter, section 7.1, describes the purposes of this particular investigation. Section 7.2 describes the reaction system in detail as well as how the data used in this investigation were obtained. Thereafter section 7.3 describes the experimental method that is used to evaluate the results obtained by the CORA algorithm as well as the results of the other algorithms considered in this investigation. Section 7.4 presents and discusses the experimental results. Finally, section 7.5 summarises the results that were obtained and presents conclusions.

The Autocatalytic problem was chosen for study for the following reasons. First, the experiments performed for chapter 6 showed that the CORA algorithm obtains more consistent results (i.e. with less variation) on problems that have a high data exemplar density, such as the Abalone problem (see section 6.3.4). The Autocatalytic data are generated synthetically and therefore as many data as required can be generated, achieving any sought after data exemplar density. It was therefore possible to generate enough data so that an adequate data exemplar density was obtained. Second, the Autocatalytic problem is in the author's opinion a relatively complicated and therefore difficult problem to solve. This is because the problem involves nonlinear time series prediction, exhibits chaotic phenomena and the data contain noise.

## 7.1 Purpose of Investigation

As described in section 5.2.2, the evaluation of the CORA algorithm is split into two parts. The first part, evaluation of the CORA algorithm and its subcomponents, is presented in chapter 6. The second part of the evaluation is presented in this chapter. The primary purpose of the investigation presented in this chapter is to determine whether the CORA algorithm is capable of constructing rule models that are better than those generated by existing techniques.

The particular CORA model properties that are evaluated in this comparison are the predictive accuracy of the model on unseen data and the model complexity. Model complexity is evaluated in terms of four different complexity measures, viz. number of rules that make up the model concept complexity, parameter complexity and the average number of rules that need to be evaluated to determine the rule model's output. The definitions of these complexity measures are identical to those given in section 5.1.1.

The secondary purpose of the investigation is to briefly study the rule models derived by the CORA algorithm, specifically the visual appearance of a fuzzy rule and the time series that is predicted by the rule model.

## 7.2 The Chaotic Reaction System

### 7.2.1 Description of the Reaction System

Grey and Scott (1983, 1984) proposed the reaction system studied here. The system is composed of two parallel, autocatalytic reactions that take place in an isothermal, continuously stirred tank reactor with catalyst decay. Lynch (1992) added a second autocatalytic reaction to the system in order to make chaotic behaviour possible. The complete reaction system proceeds according to the following reaction stoichiometry and rate equations:





The symbols A, B, C and D represent chemical species. The reaction rate  $r_i$  of chemical  $i$  is a function of a rate constant  $k_i$  and the relevant reactant reactor, or exit<sup>1</sup>, species concentrations,  $C_i$ ,  $C_j$ , etc. If it is assumed that the reactions occur in an isothermal, continuous flow stirred tank reactor, the reaction system can be described by three independent, ordinary, differential equations. In dimensionless form, the differential equations describing the reaction system are

$$\frac{dX}{d\tau} = 1 - X - Da_X XZ^2 \quad \text{Equation 7.4}$$

$$\frac{dY}{d\tau} = 1 - Y - Da_Y YZ^2 \quad \text{Equation 7.5}$$

$$\frac{dZ}{d\tau} = 1 - (1 + Da_Z)Z + \alpha_X Da_X XZ^2 + \alpha_Y Da_Y YZ^2. \quad \text{Equation 7.6}$$

$X$ ,  $Y$  and  $Z$  respectively represent the dimensionless concentration<sup>2</sup> of species A, B and C.  $Da_X$ ,  $Da_Y$  and  $Da_Z$  are respectively the Damköhler numbers for species A, B and C.  $\alpha_X$  and  $\alpha_Y$  are the ratio of the concentration of species A to that of species B, and the ratio of the concentration of species B to that of species A, respectively, in the continuously stirred tank reactor inlet stream.  $\tau$  is dimensionless time<sup>3</sup>.

## 7.2.2 Data Generation

In order to generate the data for this case study, the differential equations (equations 7.4 through 7.6) were numerically integrated using a fifth-order Runge Kutta technique with an adaptive integration step size (Gerald and Wheatley, 1989). Initial conditions of  $X = Y = Z = 0$  at  $\tau = 0$  were used.  $Da_X$ ,  $Da_Y$  and  $Da_Z$  were set to 18000, 400 and 80, in that order. Integration was performed over 100 seconds, generating a time series of roughly 10000 data exemplars. Thereafter the data were resampled using a constant sampling rate of 0.01s.

The  $X$  attribute was embedded with an embedding dimension of five and a lag of nine. This resulted in a set of five attributes, viz.  $X(\tau-36)$ ,  $X(\tau-27)$ ,  $X(\tau-18)$ ,  $X(\tau-9)$  and  $X(\tau)$ . These five

---

<sup>1</sup> In an ideal, continuously stirred tank reactor the reactor and exit concentrations of chemical species are identical (Fogler, 1992).

<sup>2</sup> The dimensional species concentrations are calculated using the equations  $X = C_A / C_{A0}$ ,  $Y = C_D / C_{D0}$ ,  $X = C_B / C_{B0}$ .  $C_{i0}$  is the reactor inlet stream concentration of chemical specie  $i$ .

<sup>3</sup>  $\tau = t \times Q/V$  where  $t$  is time,  $Q$  is the reactor feed flow rate and  $V$  is the reactor volume.

attributes are used to predict the value of  $X$  three steps into the future, viz.  $X(\tau+3)$ . In addition, noise was added to the output variable ( $X(\tau+3)$ ) to increase the difficulty of the time series prediction problem. In particular, random noise with a greatest magnitude equal to 10% the standard deviation of the  $X$  attribute was added to  $X(\tau+3)$ . The final data set therefore consists of five attributes and a single noisy output. The data set contains 9961 exemplars. The data thus obtained will henceforth be denoted the Autocatalytic data set and the entire problem will be called the Autocatalytic problem.

## 7.3 Experimental Design

This section describes the experimental method and statistical measures used to evaluate and compare the models derived by the CORA algorithm with those obtained using the other techniques considered in this chapter.

### 7.3.1 Experimental Method

The preprocessing that was performed on the Autocatalytic data was normalisation<sup>4</sup> on the entire set of data. Note that the data were not randomised because the Autocatalytic data are time series data. Randomisation would destroy any time series information contained within the data. Thereafter the data were split into three subsets, viz. training, validation and testing data sets. The first 5971 data were used for training, the following 1990 data for validation and the remaining 2000 data were used for testing purposes.

The training data were then presented to each modelling technique in turn for model generation. During this exercise, the training parameters of each technique were tuned until the best model was obtained. The measure of model quality used to find the best model was the predictive accuracy performance of the given model on the validation data set. The best training parameter settings that were found are given in appendix C. The above methodology was used for all the algorithms evaluated in this chapter with the exception of the CART algorithm (and of course the MLR algorithm). The methodology followed for the CART algorithm is as follows. First, the training and validation data were combined into a single set of data. The CART algorithm then performed ten-fold cross validation on these data, generating ten regression trees. The regression

---

<sup>4</sup> The entire set of data were normalised so that each attribute as well as the output has a mean of zero and a standard deviation of one.

tree with the overall best predictive accuracy for any of the ten folds was then exclusively used for further experimentation.

Once the best training parameters were determined for each algorithm, the training and validation data were combined into a single set of training data (with the exception of the experiments performed with the CART algorithm). Each algorithm then constructed a final model using these training data. Finally, the predictive accuracy of the generated model was determined, in a once-off fashion, on the testing data. For the CART algorithm the testing accuracy was determined using the best regression tree (as defined above). For algorithms that contain a stochastic component, three experiments using the best training parameters (obtained using the original validation data) were performed, each with a different random seed. This was done so that the variation in the results obtained by these algorithms could be calculated.

The specific manner in which the CORA algorithm was used is as follows. First, the training and validation data were used to find the best algorithmic training parameters. The knowledge of the CORA algorithm and its properties that was gained in chapter 6 was used to assist the search for the best set of training parameters. Thereafter three runs, each using a different random seed, were performed using the combined training and validation data for training. For each of the three CORA models the predictive performance of the model was then determined on the testing data.

### 7.3.2 Models and Modelling Techniques Evaluated

The following models and modelling techniques are evaluated in this chapter.

- Crisp regression trees induced by the CART algorithm. The regression trees are converted into a set of identically performing “if...then...” rules before being evaluated.
- Two radial basis function neural networks. The first is trained using the Growing Neural Gas (GNG) algorithm and the second using  $k$ -means clustering. Both networks contain one hidden layer and an output layer comprising of a single summation node. The hidden layers of both networks use Gaussian transfer functions as nodes. The trained networks are each converted to a set of 0<sup>th</sup>-order Sugeno fuzzy rules before being evaluated. The  $k$ -means radial basis function network contains an additional linear model that is converted to a linear equation to supplement the fuzzy rule model.

- A multilayer perceptron trained with the generalised delta rule using an adaptive learning rate. The perceptron contains one hidden layer and an output layer with a single output node. tanh transfer functions are used for the nodes of the network's hidden layer. The output node is a linear summation node. The neural network architecture includes a bias term.
- Two multivariate regression spline models trained by two MARS algorithm variants. The first variant, labelled MARS(nb), does not employ bootstrapping. The second variant, labelled MARS(b), does employ bootstrapping.
- A multivariate linear regression model trained using the least squares approach.
- Two fuzzy rule models generated by two different variants of the CORA algorithm. The first CORA algorithm variant, labelled CORA hereafter, is the standard algorithm described in chapter 4. The second variant, henceforth labelled CORA(mo), attempts to minimise the attribute space overlap of the fuzzy rules in an identical manner used for the overlap minimisation experiments for section 6.3.1.

### 7.3.3 Hypothesis Testing and Statistical Measures

The paired-sample t-test (Sachs, 1984) is used to determine whether the properties (e.g. predictive accuracy) of the models obtained by the various techniques are significantly different from each other. Equation 7.7 is the equation used to calculate the t-test statistic. The Bonferroni adjustment (Day and Quinn, 1989) is applied in order to minimise the probability of increases in Type I errors (Snedecor and Cochran, 1967).

$$t_2 = \frac{\bar{x}_1 - \bar{x}_2}{\sqrt{\frac{\sigma_1^2}{n_1} + \frac{\sigma_2^2}{n_2}}}. \quad \text{Equation 7.7}$$

$\bar{x}_i$  and  $\sigma_i^2$  represent the average and variance of the given property value over all  $n_i$  experiments that were performed by algorithm  $i$ . The null hypothesis (that two algorithms have the same level of, for example, predictive accuracy) is rejected with  $100(1-\alpha)\%$  confidence if

$$|t_2| \geq t_{\frac{\alpha}{2p}, r} \quad \text{where } p = \frac{k(k-1)}{2}. \quad \text{Equation 7.8}$$

In equation 7.8  $\alpha$  represents the confidence level (e.g. 0.05),  $r$  the problem degrees of freedom and  $k$  the number of replications (experiments) that were performed using a particular algorithm.

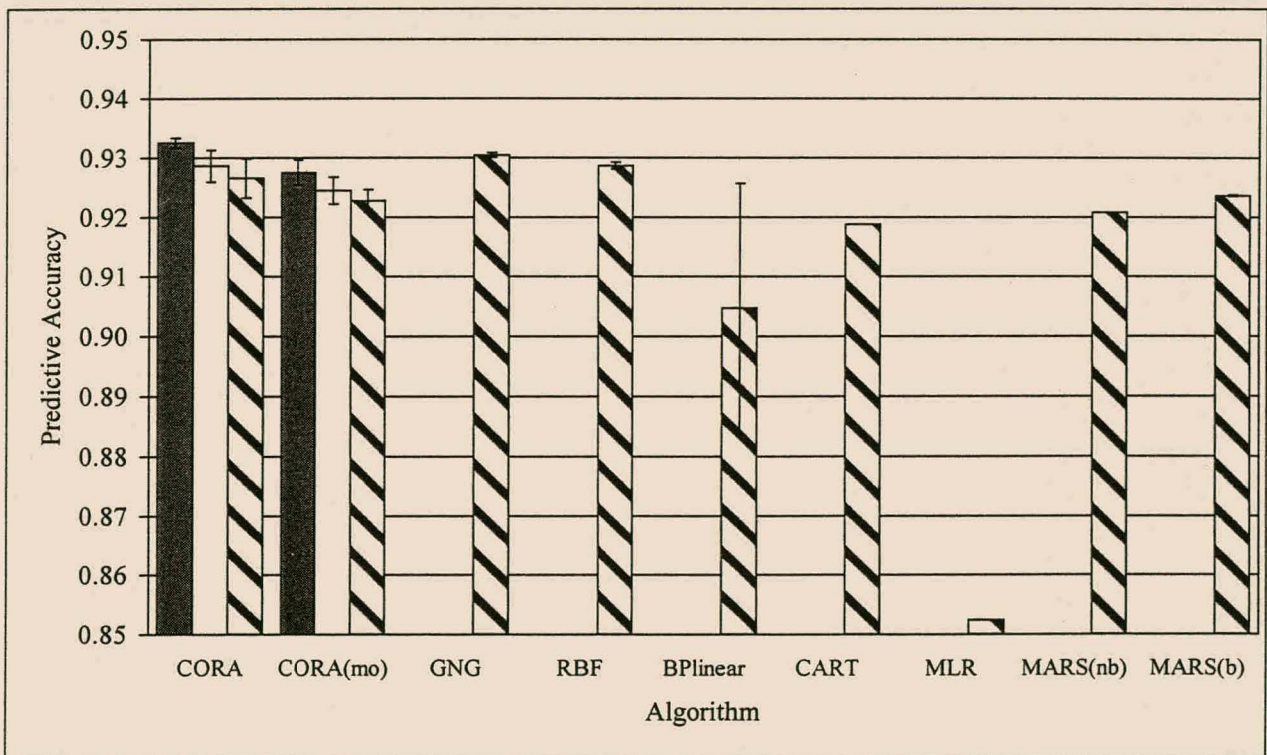


Note that this is a two-tailed test, meaning that the null hypothesis can be rejected either if the model generated by algorithm one is more accurate than the model generated by algorithm two, or vice versa.

## 7.4 Results

The results obtained for this chapter are split into three parts. Section 7.4.1 examines the predictive accuracy of the CORA models vs. the accuracy exhibited by the models of the other techniques considered in this chapter. The second part of the results, presented in section 7.4.2, presents a comparison of the complexity of the various models. Finally, section 7.4.3 examines the visual characteristics of a fuzzy rule, both before and after membership function merging. In addition, the time series prediction of the fuzzy rule model is briefly discussed in section 7.4.4.

### 7.4.1 Comparative Predictive Accuracy



**Figure 7.1 Predictive Accuracy of CORA Algorithm and Other Modelling Techniques**

Figure 7.1<sup>5</sup> compares the predictive accuracy (in terms of  $R^2$ ) on the test data obtained by the CORA models with the accuracy obtained by the models generated by each of the other techniques listed in section 7.3.2. The horizontal axis labels for each algorithm are the same as those used in chapter 6 (see section 6.1.2) with the exception of the labels for the CORA and the MARS algorithms. (The meaning of the CORA and MARS algorithm labels is explained in section 7.3.2.)

Each bar or column in the figure represents the average accuracy obtained by the given model over the three different random seed runs. For both CORA variants the predictive accuracy obtained by each of the three CORA algorithm subcomponents, viz. the reactive tabu search (RTS) component, the membership function merging (MRG) component and the rule reduction (RED) component, is reported<sup>6</sup>. The black error bar that is superimposed over each column represents the standard deviation exhibited by the given set of predictive accuracy results. The CART and MLR algorithm are deterministic and therefore only a single experiment was performed using each of these two algorithms.

Modelling Technique	CORA			CORA (minimum overlap)		
	RTS	MRG	RED	RTS	MRG	RED
GNG	✓				✗	✗
RBF	✓				✗	✗
BPlinear						
CART	✓✓	✓✓	✓	✓✓	✓✓	✓✓
MLR	✓✓	✓✓	✓✓	✓✓	✓✓	✓✓
MARS (no bootstrap)	✓✓	✓	✓	✓	✓	
MARS (bootstrap)	✓✓	✓		✓		

**Table 7.1 Predictive Accuracy Significance Results (An explanation of the meaning of the ticks and crosses in the table is given in the first paragraph of p.215.)**

<sup>5</sup> For interest sake, the so-called “no prediction accuracy” for the test data of the Autocatalytic problem is as follows. If the current value of  $X$ , viz.  $X(\tau)$ , is used to predict the next value of the  $X$  time series ( $X(\tau+1)$ ) then the one step ahead testing accuracy is  $R^2 = 0.90$ . If  $X(\tau)$  is used to predict three steps ahead (as is done in all the experiments performed for this investigation), viz.  $X(\tau+3)$ , then the testing accuracy is  $R^2 = 0.72$ .

<sup>6</sup> The solid grey column represents the average predictive accuracy obtained by the RTS component. The white column represents the average predictive accuracy obtained after the RTS and MRG components have both been employed. The striped column represents the average predictive accuracy results obtained after all three subcomponents of the CORA algorithm have been utilised.

The statistical significance of the difference in results exhibited by the various models is summarised in table 7.1. The significance of the results obtained by the GNG, RBF, BPlinear, CART, MLR, MARS(nb) (i.e. MARS with no bootstrapping) and MARS(b) (i.e. MARS with bootstrapping) are each compared with the results obtained by the RTS, MRG and RED subcomponents of the CORA algorithm (both algorithmic variants). The ✓ symbol indicates that the given CORA algorithm subcomponent constructs a model that is significantly more accurate than the model generated by the other algorithm under consideration with at least 95% but less than 99.9% confidence. The ✓✓ symbol indicates the same but with at least 99.9% confidence. No table entry indicates that neither algorithm constructs a model with predictive accuracy that is significantly different (with greater than 95% confidence) from that of the other model under consideration. The ✕ symbol indicates that the CORA model is significantly less accurate than the other model at a confidence level of at least 95%. The ✕ ✕ symbol indicates the same but with at least a 99.9% confidence level that the accuracy of the CORA model is worse than that of the other model under consideration.

From table 7.1 it can be seen that the CORA models are significantly more accurate than both the multivariate linear regression (MLR) model and the regression tree induced by CART. This phenomenon holds true no matter which CORA variant or subset of CORA algorithm components are utilised during fuzzy rule model construction. In addition, the models generated by the RTS component of the standard CORA variant are significantly more accurate than the models derived by the GNG, RBF, MARS(nb) and MARS(b) algorithms.

However, if the MRG and RED components of the CORA algorithm are employed then the rule models of the standard CORA algorithmic variant are in general (the exception is the RED rule model vs. the MARS(b) model) significantly more accurate than the MARS models only. Furthermore, for the experiments performed for this investigation the MRG and RED rule models of the minimum overlap CORA variant are found to be significantly less accurate than the GNG and RBF models.

No statistically significant difference in the results obtained by the two CORA variants and the multilayer perceptron trained by the generalised delta rule (BPlinear) are found. This is principally because of the high standard deviation of the accuracy results exhibited by the BPlinear models.

Model Property	CORA (minimum overlap) vs. CORA <sup>7</sup> (%)
Predictive Accuracy	(84)
Number of Rules	79
Number of Concepts	95
Number of Parameters	98
Attribute Space Overlap	> 99

**Table 7.2 Statistical Significance Critical Confidence Levels for the Difference in Property Values of the CORA Models<sup>8</sup>**

The critical confidence levels of the statistical significance of the difference in accuracy obtained by the models of the two CORA algorithmic variants are given in table 7.2 (in the first row). Table 7.2 shows that the predictive accuracy exhibited by each of the two CORA models is only significantly different at a low confidence level of 84%.

The remaining results presented in table 7.2 will be discussed in section 7.4.2.

### 7.4.2 Comparative Model Complexity

Algorithm	Number of Rules	Number of Concepts	Attribute Space Overlap
CORA	16.3 (1.2)	130 (16)	4.94 (0.73)
CORA (minimum overlap)	15.0 (1.0)	99.0 (11)	1.71 (0.083)
GNG	30	180	6.63 (0.050)
CART	108	215	1

**Table 7.3 Comparative Model Complexity in Terms of “if...then...” Rules, Number of Concepts and Attribute Space Overlap**

Algorithm	Number of Parameters
CORA	256 (18.0)
CORA (minimum overlap)	198 (19.1)
GNG	330
RBF	423
BPlinear	36
MLR	5
MARS (no bootstrap)	43.5 (0.00)
MARS (bootstrap)	48.2 (2.02)

**Table 7.4 Comparative Model Parameter Complexity**

<sup>7</sup> Figures in parentheses indicate that the models of the standard CORA variant exhibit better property values (accuracy, etc.) than the minimum overlap CORA variant.

<sup>8</sup> The results in table 7.2 are those exhibited by the CORA rule models after all three algorithmic components (RTS, MRG and RED) have been employed.

The complexity results exhibited by the various models are presented in tables 7.3 and 7.4. The numbers in parentheses are the standard deviations of the results of the models generated by the algorithms that contain a stochastic component. Only the rule models derived by the two CORA variants, the GNG algorithm and CART are compared with respect to number of rules, number of concepts and attribute space overlap (i.e. average number of rules that need to be evaluated to determine rule model output). This is owing to the fact that the other algorithms (RBF, BPlinear, MLR and both MARS variants) do not exclusively generate rule models. All models with the exception of the CART regression tree are compared with respect to the number of model parameters.

Modelling Technique	CORA				CORA (minimum overlap)			
	R	C	P	O	R	C	P	O
GNG	✓✓	✓	✓	✓	✓✓	✓✓	✓✓	✓✓
RBF	-	-	✓✓	-	-	-	✓✓	-
BPlinear	-	-	××	-	-	-	××	-
CART	✓✓	✓✓	-	××	✓✓	✓✓	-	××
MLR	-	-	××	-	-	-	××	-
MARS (no bootstrap)	-	-	××	-	-	-	××	-
MARS (bootstrap)	-	-	××	-	-	-	××	-

**Table 7.5 Significance of Comparative Model Complexity Results**

Table 7.5 summarises the statistical significance of the difference in complexity results obtained by the various algorithms with respect to the two CORA algorithmic variants. The symbols R, C, P and O stand for the number of rules, number of concepts, number of parameters and attribute space overlap, respectively. The definitions of the ✓, ✓✓ and ×× symbols are identical to the ones given for these symbols in section 7.4.1. The “-” symbol indicates that the significance test for the particular model complexity measure and set of models was not performed.

Inspection of table 7.5 reveals the following. First, all CORA models are significantly simpler than the GNG models (upon which they are based). This phenomenon holds true for all four model complexity measures (i.e. number of rules, number of concepts, number of parameters, attribute space overlap) considered here. In addition, the CORA models contain significantly fewer parameters than the models generated by the RBF training method. Furthermore, the CORA models are significantly less complex than the CART regression tree with respect to number of rules and number of concepts.

In contrast, the models of both CORA algorithmic variants are in terms of the parameter complexity measure significantly more complex than the BPlinear, MLR, MARS(nb) and



MARS(b) models<sup>9</sup>. Furthermore, the attribute space overlap exhibited by the CORA models is significantly worse than that of the CART regression tree.

Next, consider the complexity results for the CORA models only. The last four rows in table 7.2 give the statistical significance of the difference in complexity results obtained by the two CORA variants. Table 7.2 shows that the minimum overlap CORA variant constructs models that are significantly less complex than the models generated by the standard CORA variant in terms of three of the complexity measures. These are the number of concepts, number of parameters and attribute space overlap. The former algorithmic variant also generates a rule model that consists of fewer rules than the model generated by the latter variant but the significance level of the difference is only 79%.

The results of a combined evaluation of both the accuracy and complexity results of the CORA variant models and the GNG-generated models are summarised in table 7.6. In particular, the percentage decrease or increase (indicated by parentheses) of the CORA model property value with respect to the GNG model property value is given.

Model Property	CORA			CORA (minimum overlap)		
	RTS	MRG	RED	RTS	MRG	RED
Predictive Accuracy	(0.22)	0.20	0.42	0.32	0.64	0.83
Number of Rules	33	33	46	33	33	50
Number of Concepts	5.6	13	28	5.6	28	45
Number of Parameters	3.0	7.5	22	3.0	22	40
Attribute Space Overlap	(0.054)	2.8	26	65	66	74

**Table 7.6 Average Percentage Decrease (Increase) in CORA Model Property Value vs. GNG Models<sup>10</sup>**

Inspection of table 7.6 reveals that the worst decrease in predictive accuracy exhibited by either of the CORA models is 0.83% (obtained by the minimum overlap CORA variant). The corresponding decrease in model complexity that is achieved for this loss in predictive accuracy is at least 40%. In the author's opinion, the increase in model simplicity justifies the loss in predictive accuracy even though the t-test results indicate that this particular difference between

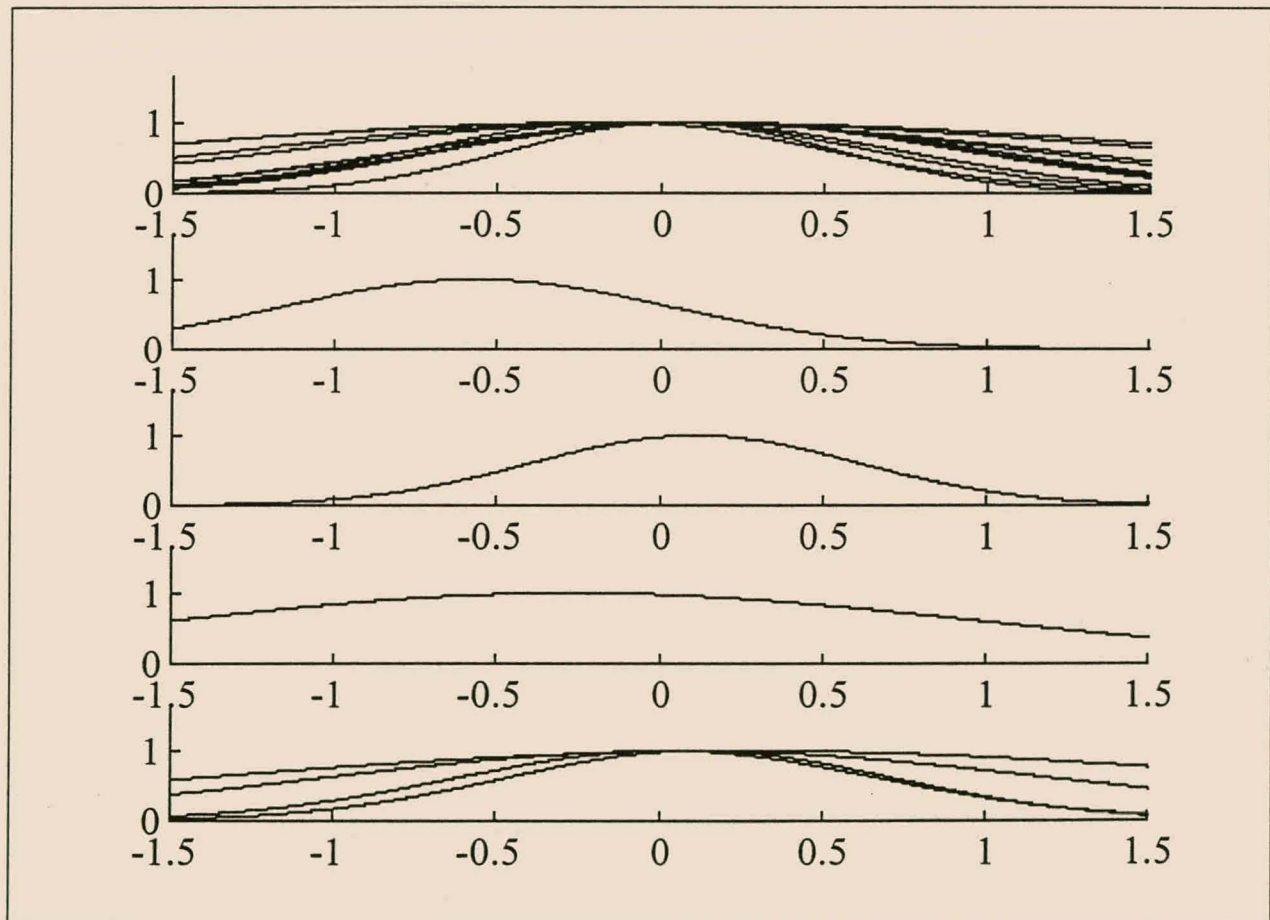
<sup>9</sup> The method used to determine the number of CORA model parameters is described in section 6.1.3.

<sup>10</sup> Percentages are calculated based on the average property values exhibited by the GNG rule models. Figures in parentheses indicate that the average property value of the CORA models is greater than the corresponding average value for the GNG models.

the predictive accuracy of the CORA and GNG models is significant at a greater than 95% confidence level (see table 7.1).

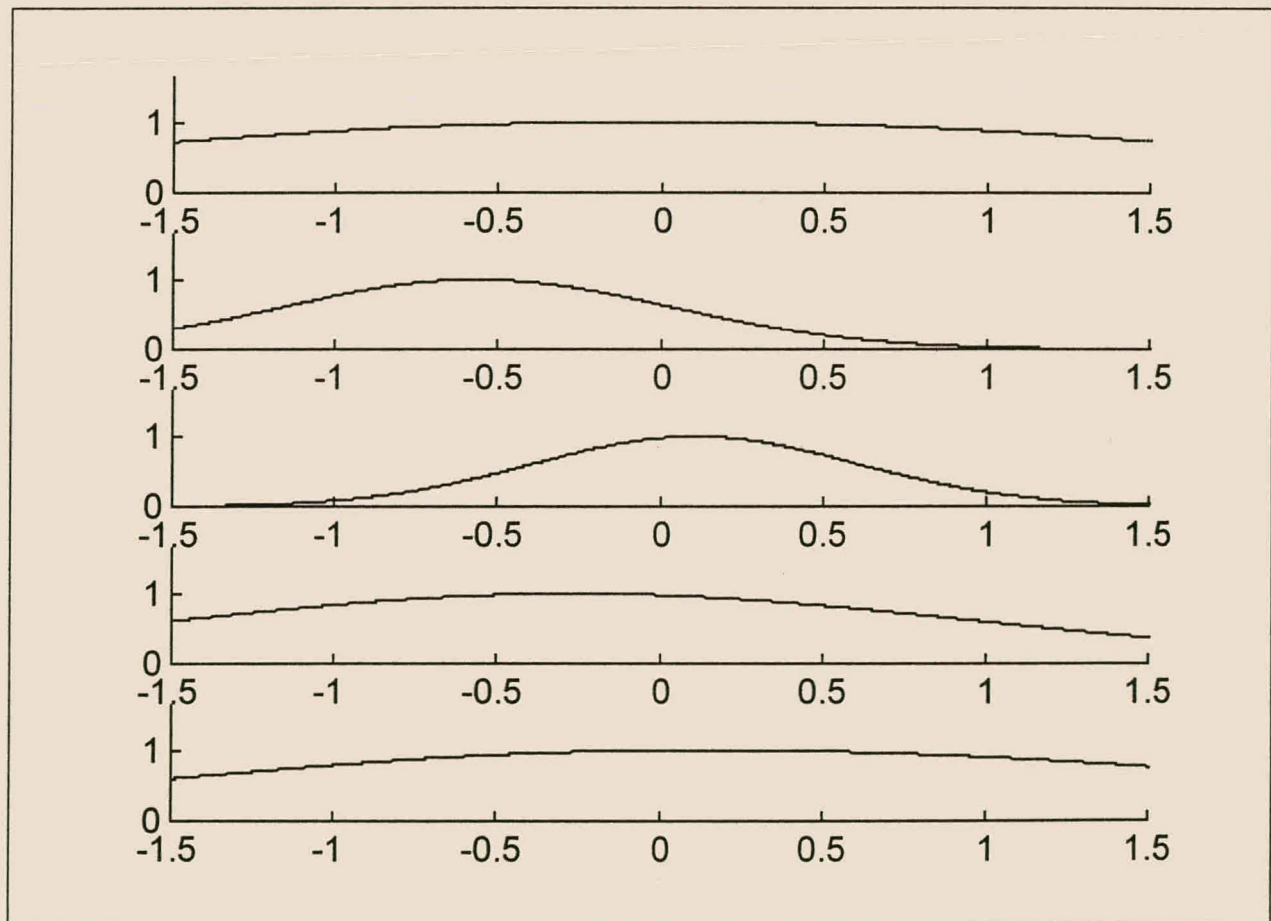
If one considers the accuracy results obtained by the standard CORA model (which were never obtained results that were significantly worse than the GNG model results at more than an 90% confidence level) then the worst drop in predictive accuracy is 0.42% (after the RED component has been applied). Unlike the above results for the minimum overlap CORA variant this drop in predictive accuracy was not found to be significant with 95% or more confidence. (The actual critical confidence level is 89%.) Furthermore, this less than 0.5% drop in predictive accuracy is accompanied by at least a 22% decrease in model complexity. In the author's opinion, the increase in model simplicity again justifies the nonsignificant (at even a 90% confidence level) loss in predictive accuracy.

### 7.4.3 Interpretation of a CORA Fuzzy Rule



**Figure 7.2 CORA (Minimum Overlap Variant) Fuzzy Rule Assembled by the RTS Component**

Figures 7.2 and 7.3 present examples of fuzzy rules generated by the CORA algorithm. In particular, the membership functions of these rules are shown. The consequent for the fuzzy rule in figure 7.2 is  $-1.1$  and the consequent for the fuzzy rule in figure 7.3 is  $-0.82$ . A negative consequent means that the rule will exhibit an inverse response to any data exemplar presented to it. In each of these figures the membership functions of the fuzzy rule are grouped according to each dimension or attribute of the Autocatalytic data. For example, the membership functions that pertain to the first attribute, viz.  $X(\tau-36)$ , are given in the top subplot of a figure. The rule in figure 7.2 was generated by the RTS component of the minimum overlap CORA algorithmic variant. No merging has been performed on the rule. Figure 7.3 presents the same rule but after membership function merging has been performed. The merging threshold, obtained using the AIC and BIC information criteria, was 0.996. (See section 4.3.4.4 for details regarding the membership function merging threshold and the two information criteria.)



**Figure 7.3 CORA (Minimum Overlap Variant) Fuzzy Rule Obtained after Membership Function Merging<sup>11</sup>**

<sup>11</sup> The merging threshold used for this particular merging exercise was 0.996.



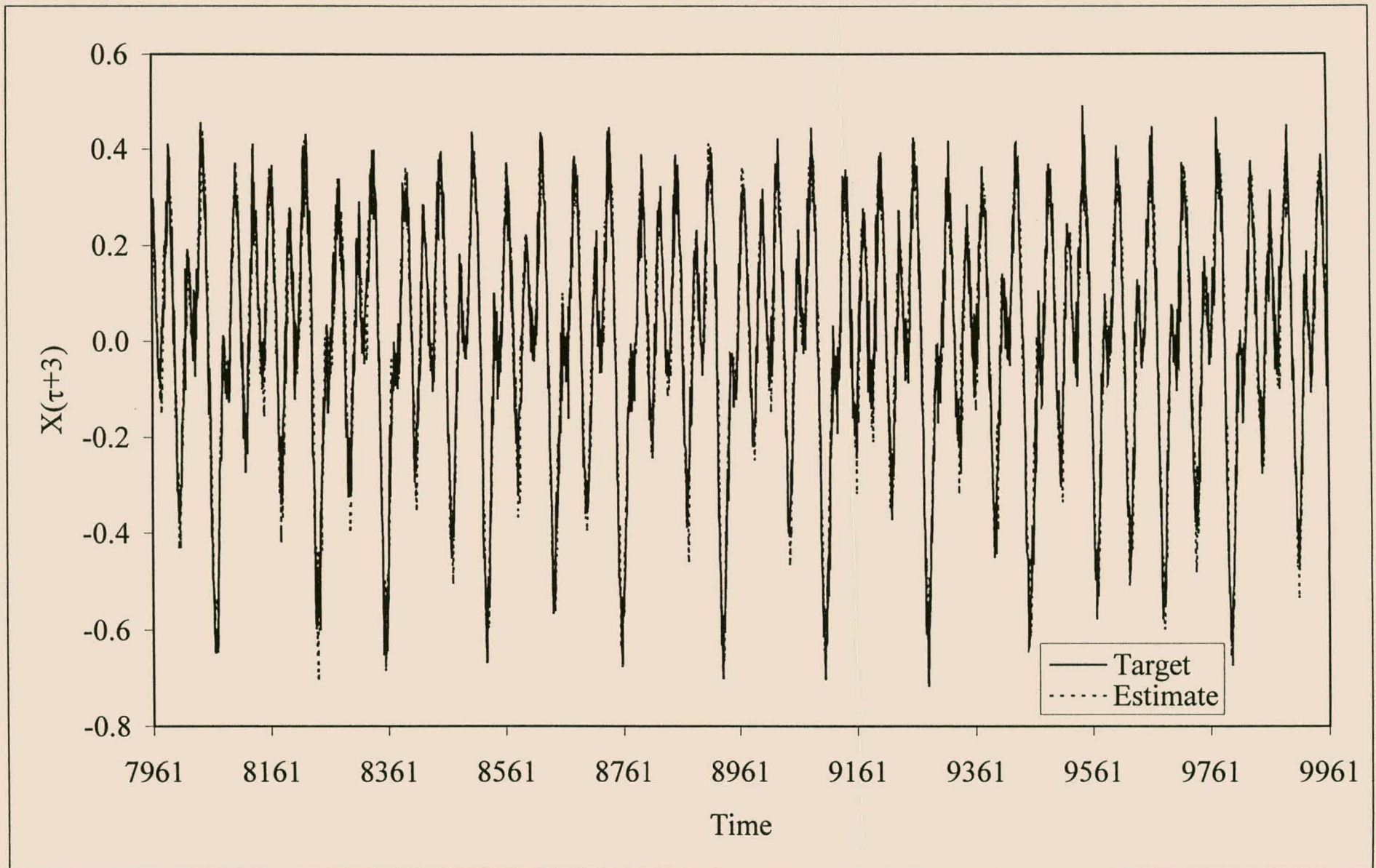
Two interesting phenomena can be observed in figures 7.2 and 7.3. First, it is clear that for this particular fuzzy rule membership function merging appreciably simplifies the antecedents pertaining to the  $X(\tau-36)$  and  $X(\tau)$  attributes. In particular, the number of membership functions used for the  $X(\tau-36)$  attribute is reduced from nine to one. In addition, the number of membership functions for the  $X(\tau)$  attribute is reduced fourfold. For this particular rule such membership function merging causes the antecedent part of the merged rule to contain no fuzzy disjunction operations anymore. The simplified rule can be rewritten as

**If  $X(\tau-36)$  is *Fairly Medium* and  $X(\tau-27)$  is *Low* and  $X(\tau-18)$  is *Medium* and  $X(\tau-9)$  is *Medium-Low* and  $X(\tau)$  is *Medium-High* then  $X(\tau+3)$  is  $-0.82$ .**

Second, most of the membership functions depicted in figures 7.2 and 7.3 are fairly wide in relation to the attribute ranges. (The attribute values of the Autocatalytic problem range between  $-0.75$  and  $+0.75$ .) This can be attributed to the fact that the Autocatalytic problem is a fairly high-dimensional problem. The Growing Neural Gas algorithm, from which the original membership functions are obtained, is forced to use wide membership functions in order to be able to adequately cover the high-dimensional attribute space. Such wide membership functions cause the set of fuzzy rules to form a smooth output prediction surface. This in turn usually improves the generalisation capabilities (on previously unseen data) of the fuzzy rule set.

#### 7.4.4 Fuzzy Rule Model Time Series Prediction

Figure 7.4 presents the testing estimated output of the fuzzy rule model vs. the target  $X(\tau+3)$  time series. The results are taken from one of the testing experimental runs performed with the standard CORA algorithmic variant. It can be seen that the predicted output of the fuzzy rule model follows the target time series fairly closely over the entire time span, even though the target time series is sharply peaked. However, close inspection reveals that the estimated time series more often than not does not precisely follow the peaks of the true time series. The reason for this is that consequent magnitude penalisation was employed during the experimental run. As described in section 4.3.4.6, increased penalisation causes the model output prediction surface to become increasingly smooth. This seems to have happened here to such a degree that the CORA model was unable to exactly match the largest peaks or deviations of the Autocatalytic time series.



**Figure 7.4** Estimated vs. Target Autocatalytic Times Series



## 7.5 Summary and Conclusions

In summary, the standard CORA variant in its entirety (employing all algorithmic components) generally obtains significantly more accurate models than those generated by the CART, MLR and MARS algorithms (the baseline confidence level is 95%). However, only the RTS models are significantly more accurate than the GNG and RBF models.

The models of the complete minimum overlap CORA algorithmic variant are also significantly more accurate than those obtained using MLR or CART. In addition, the models generated by the RTS component of the minimum overlap CORA variant are generally significantly more accurate than the models of either of the MARS variants. However, after either the MRG or both the MRG and RED components have been employed the resultant rule models are significantly less accurate than both the GNG and RBF models.

No significant conclusions can be reached regarding the comparative accuracy performance of the BPlinear and CORA models because of the high variance of the accuracy results of the BPlinear models.

In terms of model complexity, all CORA models are significantly simpler than the GNG models upon which they are based. In addition, the CORA models contain significantly fewer parameters than the RBF models and less rules and concepts than the CART regression tree. In contrast, the models of both CORA algorithmic variants contain significantly more parameters than the BPlinear, MLR, MARS(nb) and MARS(b) models. Furthermore, the attribute space overlap of the CORA models is significantly worse than that of the CART regression tree.

Inspection of the accuracy and complexity results of the GNG and CORA models brought to light the following. The worst percentage drop in average predictive accuracy (in comparison to the GNG results) exhibited by the standard CORA models is 0.42%. The corresponding decrease in model complexity is at least 22%. Furthermore, the worst decrease in predictive accuracy exhibited by the minimum overlap CORA models is 0.83% with a corresponding decrease in model complexity of at least 40%. In the author's opinion, the increase in model simplicity justifies the loss in predictive accuracy, even though for the latter CORA variant the difference in the accuracy results between the CORA and GNG models is statistically significant.

In conclusion, for the Autocatalytic problem studied in this chapter, the CORA algorithm is capable of building models with either better or comparable accuracy to the models constructed by the other modelling techniques that were evaluated. This means that it is not always necessary

to use additional mathematical constructs (as does for instance the RBF technique) to improve model predictive performance. Rather, these results indicate that improved predictive performance can be achieved with improved learning methods and the intelligent application of natural language constructs (such as fuzzy conjunction and disjunction operators) in the antecedent parts of fuzzy rules. In addition, the intelligent use of such fuzzy operators can significantly reduce model complexity for (in the author's opinion) a relatively small drop in predictive accuracy. Lastly, the successful application of the CORA algorithm to the modelling of the Autocatalytic problem shows that it is possible to construct relatively intelligible "if...then..." fuzzy rule models from chemical process data that are competitive (in terms of accuracy) with other model types (e.g. multivariate spline models).

# Chapter 8

## Conclusions and Future Work

The artificial intelligence community has developed a large body of algorithms that can be employed as powerful data analysis tools. However, discussions with petrochemical plant process engineers (section 1.1) brought to light that such techniques are currently not readily used in operational decision support. The primary reasons for this are as follows. First, process operator personnel usually have little secondary and no tertiary education. Intelligibility and comprehensibility are therefore important requirements for any process model. However, existing techniques (e.g. the CART regression tree algorithm) that construct relatively intelligible models do not meet the necessary accuracy requirements. Conversely, the process operators and even the process engineers generally find that the models generated by techniques (e.g. neural networks) that do meet the accuracy requirements are difficult to understand and to interpret. This latter factor often leads to the eventual abandonment of decision support tools that are based on relatively unintelligible process models.

### 8.1 Evaluation of Existing Rule Building Techniques

In the light of these findings, an analysis of the limitations of existing techniques that construct intelligible models was performed. “if...then...” rule-based models were taken to be of adequate intelligibility, based on the comprehensibility postulate of Michalski (1983) as well as the results of comparative studies available in the literature. The analysis therefore focussed on algorithms that construct rule-based models. The particular characteristics of existing techniques that were

investigated are the search strategies used to construct rules and the historical trends in rule format. The investigation revealed the following.

- ***Nongreedy vs. greedy.*** Techniques that are not completely greedy and that are able to explicitly detect and utilise attribute interactions (typical in most chemical processes) during rule construction can produce rule models that are significantly less complex, and therefore more intelligible, than those obtained using greedy techniques. Furthermore, in most cases such rule sets have predictive accuracy that is comparable to the accuracy of rule sets obtained with greedy methods. A major problem with nongreedy search methods is the significant increase in computational complexity incurred by using such methods.
- ***Data fragmentation and concept replication.*** Training data set fragmentation and concept replication during rule construction can have a significantly detrimental effect on model predictive accuracy and comprehensibility. This is true for most decision tree induction algorithms as well as for set covering methods that employ a “divide and conquer” approach during rule construction. Empirical results indicate that algorithms that do not suffer from training set fragmentation can build rule models that are significantly more accurate and in some cases less complex than models obtained using techniques that do have this problem.
- ***Rule format.*** The historical trend in both decision tree and fuzzy rule format has been towards increased mathematical content. The primary aim of this change in model format is to improve model predictive accuracy. In the author’s opinion, the improvements in predictive accuracy usually do not justify the decrease in model comprehensibility and intelligibility.
- ***Internal disjunction.*** Rule set complexity can be further reduced if internal disjunction in the antecedent part is allowed and the search for such rules is guided using intelligent search restrictions. Such search restrictions can also significantly reduce the computational complexity involved in finding internally disjunctive rules. As far as the author is aware, internal disjunction is generally not used in fuzzy modelling as a means of improving accuracy rather than resorting to the use of mathematical constructs in fuzzy rules.

Thereafter, based on the above conclusions, a novel fuzzy rule construction technique was proposed called the Combinatorial Rule Assembly (CORA) algorithm. The CORA algorithm constructs internally disjunctive 0<sup>th</sup>-order Sugeno rules. Rule construction proceeds by first using the Reactive Tabu Search (RTS) combinatorial search method to optimally allocate a set of membership functions generated by the Growing Neural Gas (GNG) radial basis function network training algorithm between a fixed number of fuzzy rules. The radial basis function adjacency matrix generated by the GNG algorithm is used to reduce the combinatorial search space and to bias the construction of rules towards rules that each comprises of a set of highly overlapping membership functions. After initial rule assembly, rule model simplification takes place. First, the membership functions of each rule that overlap significantly are merged into single membership functions. Second, rules are discarded that do not significantly contribute to the overall rule model performance.

## 8.2 Evaluation of the CORA Algorithm

The CORA algorithm was evaluated in two parts. The aim of first part was to determine whether the algorithm is capable of constructing rule models that are competitive with those generated by other algorithms, in terms of both predictive accuracy and model complexity. In addition, the influence of CORA training parameter settings was also investigated. The conclusions that were reached for the first part of the evaluation are the following.

- ***Geometrical form retention.*** For the Spiral problem the CORA model retains the geometrical form and training accuracy of the GNG model upon which it is based while using fewer fuzzy rules (30 instead of 105).
- ***Predictive accuracy and model complexity.*** The CORA algorithm is capable of constructing rule models with comparable or nominally better predictive accuracy than all the other modelling techniques considered, with the exception of the two multilayer perceptron variants. However, the CORA algorithm performs poorly in comparison to the other techniques that were evaluated on problems that are predominantly discrete and contain few data.

For eight of the nine problems that were studied, the generated CORA models are more accurate than the GNG models upon which the CORA models are based. The CORA rule models are less complex than their GNG counterparts in terms of number of fuzzy rules, concept complexity and parameter complexity.



For all of the nine problems, the CORA models contain more concepts than their CART and BEXA counterparts. However, in some cases the CORA models contain fewer rules, but exhibit comparable or better predictive accuracy. This means that for some problems the CORA rules have greater modelling capability than the rules used by CART or BEXA.

The CORA rule models generally contain more parameters than the models derived by the other techniques (not the GNG algorithm).

- ***Attribute space overlap.*** The CORA algorithm is able to generate models that have less attribute space overlap than their GNG-derived counterparts. However, for the Housing, Servo, Ionosphere, Slugflow and Pima problems and certain CORA training parameter settings the reduction in attribute space overlap is detrimental to model predictive performance. Such models exhibit less overlap but also have poorer predictive accuracy than their GNG-derived counterparts.

Further experimentation brought to light that for the Abalone, Auto and Housing problems and certain CORA training parameter settings the magnitude of attribute space overlap is reduced by more than two-thirds (to between 1.6 and 2.6), in comparison to the overlap of the GNG rule models without appreciably affecting model predictive accuracy. The CORA algorithm can therefore build models with attribute space overlap that approaches that of the CART and BEXA rule models. This can be done even though the GNG model (upon which the CORA models are based) overlap ranges between 4.9 and 9.1 for the same three problems.

- ***Number of assembled rules.*** For four of the five regression problems (the Abalone, Auto, Housing and Servo problems) that were studied, the correct choice of the number of assembled rules allows the CORA algorithm to build models with nominally superior predictive accuracy in comparison to the GNG-derived models. Furthermore, an increase in the number of assembled fuzzy rules generally leads to an increase in predictive accuracy. However, the use of too many rules leads to overfitting.
- ***Membership function merging.*** In the “number of fuzzy rules” experiments performed for section 6.2.1, membership function merging generally has a negative effect on predictive performance. However, for the Auto and Ionosphere problems membership function merging decreases model complexity but increases predictive

accuracy. Furthermore, for the Abalone, Housing and Pima problems merging on average and at worst decreases predictive accuracy by 1.6% for a corresponding 31% decrease in model concept complexity. In the author's opinion the drop in predictive accuracy is therefore in most cases outweighed by the increase in model simplicity.

For the experiments performed in section 6.2.1, rule reduction increased with increasing number of assembled rules. Furthermore, rule reduction, in contrast to membership function merging, does not always decrease model predictive accuracy. At worst (for the Ionosphere classification problem using thirty rules) rule reduction decreased predictive accuracy by 0.87%. However, rule reduction is generally able to appreciably decrease model complexity. For the problems considered in section 6.2.1, model complexity decreased by on average 19% in terms of model concepts and by 22% in terms of model parameters.

- ***Swap (or move) threshold.*** An increase in the swap (or move) threshold employed by the CORA algorithm does not consistently lead to significantly different (more or less accurate, or less complex) rule models. This means that a relatively stringent swap (or move) threshold value such as 0.9 can be used to substantially decrease computational complexity without seemingly altering the properties of the models that are derived. However, it seems that the adaptive modification of the search resolution by the RTS component, from an initial value of one percent up to the swap (or move) threshold, plays a more significant role in determining final rule model properties than the minimum swap (or move) threshold.
- ***Swaps vs. moves.*** The use of moves or both swaps and moves by the RTS component, as a means of assembling fuzzy rules, instead of exclusively using swaps does not seem to have a consistently beneficial or detrimental effect on model predictive accuracy. However, the use of moves rather than swaps generally leads to greater model complexity reduction by the MRG (membership function merging) and RED (rule reduction) components of the CORA algorithm.
- ***Consequent magnitude penalisation.*** The use of consequent magnitude penalisation generally improves rule model predictive accuracy, especially if many fuzzy rules are assembled. In addition, consequent magnitude penalisation reduces the variance of the RTS component predictive accuracy results for four of the six problems studied in section 6.2.4. This generally leads to reduced variation in the predictive

accuracy results of the MRG and RED components. Furthermore, in some cases the use of consequent magnitude penalisation leads to greater rule model simplification.

- **Fuzzy operators.** On the Auto problem the use of Yu fuzzy operators rather than Zadeh fuzzy operators allows the CORA algorithm to construct more accurate rule models. However, greater model simplification is possible if Zadeh operators are used. In the author's opinion, the average gain in predictive accuracy (2.1%) outweighs the average increase in model concept complexity (6.6%) if Yu, rather than Zadeh operators are used.
- **Combinatorial search complexity.** The use of the GNG adjacency matrix significantly reduces the complexity of the combinatorial search performed by the RTS component of the CORA algorithm. The combinatorial complexity of evaluating each rule model that can be generated using any one of all permissible swaps (or moves) is significantly less than the worst case scenario described in section 4.3.4.8 (section 4.3.4.9 for moves).
- **Local optima.** Experiments on the Abalone, Auto, Housing and Servo problems showed that for all four problems the RTS component encountered a number of local optima during its search for the rule model with the best fitness. This means that nongreedy search will obtain better solutions than simple greedy search will for these problems.
- **Variance of results.** In many experiments the models generated by the CORA algorithm using different random seeds only, produced results that exhibited greater than expected variation. It is hypothesised that there are three reasons for this phenomenon. First, the CORA algorithm struggles to model problems that have a low exemplar to attributes ratio. However, this problem is not restricted to the CORA algorithm. The same phenomenon occurs using other advanced rule construction techniques. Second, the exclusive use of the AIC and BIC information criteria by the MRG component can cause membership function merging to be based on different local optima found in the AIC and BIC results. (Both the AIC and BIC information criteria generate results that exhibit a number of local optima during the determination of the optimal level of membership function merging.) If this occurs then membership function merging will be performed at varying thresholds leading to increased variation in results. Third, the combinatorial complexity of constructing

rules using methodology of the CORA algorithm is large. It is therefore difficult for the CORA algorithm to consistently find the same solution if different random seeds are used.

The purpose of the second part of the evaluation was to find out whether the CORA algorithm can construct models that are significantly more accurate and / or significantly less complex than other modelling techniques. The problem investigated in this evaluation is the time series prediction (three steps ahead) of a noisy, nonlinear, chaotic reaction system. The experiments performed on these data revealed the following.

- **Accuracy.** The standard CORA algorithm in its entirety (employing all algorithmic components) generally obtains significantly more accurate models than those generated by the CART, MLR and MARS algorithms (the baseline confidence level is 95%). However, only the RTS models are significantly more accurate than the GNG and RBF models.

The models of the complete minimum overlap CORA variant are also significantly more accurate than those obtained using MLR or CART. In addition, the models generated by the RTS component of the minimum overlap CORA variant are generally significantly more accurate than the models of either of the MARS variants. However, after either the MRG or both the MRG and RED components have been employed the resultant rule models are significantly less accurate than both the GNG and RBF models.

- **Complexity.** All CORA models are significantly simpler than the GNG models upon which they are based. In addition, the CORA models contain significantly fewer parameters than the RBF models and significantly less rules and concepts than the CART regression tree. In contrast, the models of both CORA algorithmic variants contain significantly more parameters than the BPlinear, MLR, MARS(nb) and MARS(b) models. Also, the attribute space overlap of the CORA models is significantly worse than that of the CART regression tree.
- **Loss of accuracy vs. increase in simplicity.** Taking all results into account, the worst percentage drop in average predictive accuracy (in comparison to the GNG results) exhibited by the standard CORA models is 0.42%. The corresponding decrease in model complexity is at least 22%. Furthermore, the worst decrease in predictive accuracy exhibited by the minimum overlap CORA models is 0.83% with a

corresponding decrease in model complexity of at least 40%. In the author's opinion, the increase in model simplicity justifies the loss in predictive accuracy, even though for the latter CORA variant the difference in the accuracy results between the CORA and GNG models is statistically significant.

In conclusion, for the Autocatalytic problem the CORA algorithm can build models that have either better or comparable accuracy in comparison to the models constructed using the other modelling techniques that were evaluated. This means that it is not always necessary to use additional mathematical constructs to improve model predictive performance. Rather, improved predictive performance can be achieved using better learning methods and the intelligent application of natural language constructs (such as fuzzy conjunction and disjunction operators) in the antecedent parts of fuzzy rules. In addition, the intelligent use of such fuzzy operators can significantly reduce model complexity for (in the author's opinion) a relatively small drop in predictive accuracy. Lastly, the successful application of the CORA algorithm to the modelling of the Autocatalytic problem shows that it is possible to construct relatively intelligible "if...then..." fuzzy rule models from chemical process data that are competitive (in terms of accuracy) with other model types (e.g. multivariate spline models).

## 8.3 Future Work

The evaluation of the CORA algorithm brought to light a number of problems or limitations associated with the CORA algorithm. The proposed research tasks below are designed to address some of the more important and interesting of these problems and limitations.

- **Attribute space overlap.** In the experiments performed for this dissertation the attribute space overlap exhibited by the GNG models is in most cases high. High attribute space overlap makes a rule model less intelligible. The method used by the GNG algorithm to determine the width of a radial basis function directly affects the attribute space overlap of the GNG rule model and indirectly that of the CORA model. It would be desirable to reformulate the GNG width-updating procedure to reduce rule overlap and thus improve rule model comprehensibility.
- **Use of membership functions.** The current version of the CORA algorithm utilises all the membership functions generated by the GNG algorithm to construct rules. Arguably this is unnecessary. It would be interesting to study ways in which membership functions could be selectively utilised during fuzzy rule assembly. For



instance, a dummy rule could be created. Such a rule would take part in the membership function swapping or moving operations of the RTS component but be ignored during rule consequent identification as well as rule model fitness evaluation. The dummy rule would then act as a repository of membership functions that can be selectively utilised by the RTS component to build better rules.

- ***Combinatorial search focus.*** Currently the RTS component is allowed to change the antecedent part of each rule in the rule model. During the experiments performed for this dissertation it was found that some rules contributed more than others towards the quality (e.g. predictive accuracy, attribute space overlap) of the rule model. It would be interesting to study whether the RTS search can be reformulated to only attempt to improve poorly performing rules. This would lead to a significant reduction in the size of the combinatorial search and would also explicitly focus the RTS search towards improving the worst performing rules.
- ***Membership function merging.*** Another manner in which the computational expense of rule assembly can be reduced is to perform membership function merging during antecedent part construction by the RTS component, rather than in a once-off fashion after the RTS search has been completed. In particular, membership function can be performed once a reasonable rule model has been constructed. The merged set of membership functions can then be used to again build a rule model, and so on.
- ***“Right-sized” model determination.*** It was determined in chapter 6 that the AIC and BIC information criteria that are currently used to determine the “right-sized” GNG model as well as the level of membership function merging are suboptimal. As mentioned in chapter 6, there are a number of other measures that can be used to find the “right-sized” model, etc. An example of these is the minimum description length measure (see e.g. Quinlan and Rivest, 1989). It would be instructive to see whether the CORA algorithm produces better and more consistent models (less variance in results) based on one or more of these measures.
- ***Choice of data.*** Results in chapter 3 and chapter 6 showed that it is often the case that advanced rule construction techniques struggle to perform better than simple greedy techniques on problems which exhibit a low data exemplar density. It is suggested that this phenomenon be studied more comprehensively and in more detail to determine the extent and generality of this problem.

# References

- Abarbanel, H.D.I. *Analysis of observed chaotic data*, 1996. Springer, New York.
- Abe, S. and Lan, M.-S. A method for fuzzy rules extraction directly from numerical data and its application to pattern classification. *IEEE Transactions on Fuzzy Systems*, 3(1):18-28, February 1995.
- Abe, S. and Thawonmas, R. A fuzzy classifier with ellipsoidal regions. *IEEE Transactions on Fuzzy Systems*, 5(3):358-368, August 1997.
- Aha, D.W., Kibler, D. and Albert, M.K. Instance-based learning algorithms. *Machine Learning*, 6:37-66, 1991.
- Aha, D.W. and Salzberg, S.L. Learning to catch: applying nearest neighbour algorithms to dynamic control tasks. In *Selecting Models from Data: AI and Statistics IV; Lecture Notes in Statistics*, 89:321-328, 1994. Springer-Verlag, New York, Cheeseman, P. and Oldford, R.W. (eds.).
- Akaike, H. Statistical predictor identification. *Annals of the Institute for Statistical Mathematics*, 22:203-217, 1970.
- Akaike, H. Information theory and an extension of the maximum likelihood principle. In *Proceedings of the Second International Symposium on Information Theory*, 267-281, Akademia Kiado, Budapest, 1973.
- Akaike, H. A new look at statistical model identification. *IEEE Transactions on Automatic Control*, 19:716-723, 1974.
- Akaike, H. On entropy maximization principle. In *Applications of Statistics*, 27-41, 1977. North-Holland Publishing Company, Krishnaiah, P. (ed.).
- Alberts, H.A. Personal communication, January 1997.
- Andrews, R., Diederich, J. and Tickle, A.B. Survey and critique of techniques for extracting rules from trained artificial neural networks. *Knowledge-based Systems*, 8(6):373-383, 1995.
- Arciszewski, T., Michalski, R.S., Wnek, J. Constructive induction: the key to design creativity. In *Proceedings of the Third International Round-Table Conference on Computational Models of Creative Design*, 397-426, Heron Island, Queensland, Australia, December 3-7, 1995.
- Back, D. and Hoffmeister, F. Extended selection methods for genetic algorithms. In *Proceedings of the First International Conference on Genetic Algorithms*, San Mateo, California, 1991.

- Baldwin, J.F., Lawry, J. and Martin, T.P. Mass assignment based induction of decision trees on words. *International Journal of Intelligent Systems*, 12(7):523-548, 1997.
- Battiti, R. Reactive search: towards self-tuning heuristics. In *Modern Heuristic Search Methods*, 61-83, 1996. John Wiley and Sons, New York, Rayward-Smith, V.J., Osman, I.H., Reeves, C.R. and Smith, G.D. (eds.).
- Battiti, R. and Tecchiolli, G. The reactive tabu search. *ORSA Journal on Computing*, 6(2):126-140, 1994a.
- Battiti, R. and Tecchiolli, G. Simulated annealing and tabu search in the long run: a comparison on QAP tasks. *Computers and Mathematics with Applications*, 28(6):1-8, 1994b.
- Battiti, R. and Tecchiolli, G. Local search with memory: benchmarking RTS. *Operations Research Spektrum*, 17(2/3):67-86, 1995a.
- Battiti, R. and Tecchiolli, G. Training neural nets with the reactive tabu search. *IEEE Transactions on Neural Networks*, 6(5):1185-1200, 1995b.
- Beasley, J.E. and Chu, P.C. A genetic algorithm for the set covering problem. *European Journal of Operational Research*, 94:392-404, 1996.
- Bellman, R.E. *Adaptive Control Processes*, 1961. Princeton University Press.
- Bennett, K.P. and Mangasarian, O.L. Robust linear programming discrimination of two linearly inseparable sets. *Optimization Methods and Software*, 1:23-34, 1992.
- Berardinis, L.A. Clear thinking on fuzzy logic. *Machine Design*, 64(8):46-52, April 1992.
- Berlich, R., Kunze, M. and Steffens, J. A comparison between the performance of feed forward neural networks and the supervised growing neural gas algorithm. In *Proceedings of the Fifth Workshop on Artificial Intelligence in High Energy Physics*, Lausanne, 1996. World Scientific.
- Bezdek, J.C. *Fuzzy Mathematics in Pattern Classification*. Ph.D. dissertation, Applied Mathematics Center, Cornell University, Ithaca, 1973.
- Billings, S.A. and Hong, X. Dual-orthogonal radial basis function networks for nonlinear time series prediction. *Neural Networks*, 11:479-493, 1998.
- Blake, C., Keogh, E. and Merz, C.J. UCI Repository of machine learning databases [<http://www.ics.uci.edu/~mllearn/MLRepository.html>]. University of California, Irvine, California, 1998.
- Bloedorn, E. and Michalski, R.S. Data-driven constructive induction in AQ17-PRE: a method and experiments. In *Proceedings of the IEEE International Conference on Tools for AI*, 27-30, San Jose, California, November 1991.
- Bloedorn, E. and Michalski, R.S. The AQ17-DCI system for data-driven constructive induction and its application to the analysis of world economics. In *Proceedings of the Ninth International Symposium on Methodologies for Intelligent Systems (ISMIS-96)*, 108-117, Zakopane, Poland, June 10-13, 1996.
- Box, G.E.P. and Jenkins, G.M. *Time Series Analysis: Forecasting and Control*, 1970. Holden-Day, San Francisco, California.
- Breiman, L. Bagging predictors. *Machine Learning*, 24:123-140, 1996.

- Breiman, L., Friedman, J.H., Olshen, R.A. and Stone, C.J. *Classification and Regression Trees*, 1984. Chapman & Hall, International Thomson Publishing, New York.
- Brodley, C.E. Recursive automatic bias selection for classifier construction. *Machine Learning*, 20:63-94, 1995.
- Brodley, C.E. and Utgoff, P.E. Multivariate decision trees. *Machine Learning*, 19:45-77, 1995.
- Brown, D.E., Pittard, C.L. and Han, P. Classification trees with optimal multivariate decision nodes. *Pattern Recognition Letters*, 17(7):699-703, 1996.
- Broomhead, D.S. and Lowe, D. Multivariable functional interpolation and adaptive networks. *Complex Systems*, 2:321-355, 1988.
- Bruske, J. and Sommer, G. Dynamic cell structures. In *Advances in Neural Information Processing Systems*, Volume 7, 497-504, 1995a. MIT Press, Cambridge, Massachusetts, Tesauro, G., Touretzky, D.S. and Leen, T.K. (eds.).
- Bruske, J. and Sommer, G. Dynamic cell structure learns perfectly topology preserving map. *Neural Computation*, 7(4):845-865, 1995b.
- Cai, L.Y. and Kwan, H.K. Fuzzy classifications using fuzzy inference networks. *IEEE Transactions on Systems, Man, and Cybernetics – Part B*, 28(3):334-347, June 1998.
- Carpenter, G.A., Grossberg, S. and Reynolds, J.H. ARTMAP: supervised real-time learning and classification of nonstationary data by a self-organizing neural network. *Neural Networks*, 4:565-588, 1991.
- Carpenter, G.A., Grossberg, S., Markuzon, N., Reynolds, J.H. and Rosen, D.B. Fuzzy ARTMAP: a neural network architecture for incremental supervised learning of analog multidimensional maps. *IEEE Transactions on Neural Networks*, 3(5):698-713, September 1992.
- Castro, J.L. and Zurita, J.M. An inductive learning algorithm in fuzzy systems. *Fuzzy Sets and Systems*, 89:193-203, 1997.
- Chen, J-Q. and Xi, Y.-G. Nonlinear system modeling by competitive learning and adaptive fuzzy inference system. *IEEE Transactions on Systems, Man, and Cybernetics – Part C: Applications and Reviews*, 28(2):231-238, May 1998.
- Chi, Z. and Yan, H. ID3-derived fuzzy rules and optimized defuzzification for handwritten numeral recognition. *IEEE Transactions on Fuzzy Systems*, 4(1):24-31, February 1996.
- Cho, K.B. and Wang, B.H. Radial basis function based adaptive fuzzy systems and their applications to system identification and prediction. *Fuzzy Sets and Systems*, 83:325-339, 1996.
- Choi, C.H. and Choi, J.Y. Constructive neural networks with piecewise interpolation capabilities for function approximations. *IEEE Transactions on Neural Networks*, 5:255-266, 1994.
- Cios, K.J. and Sztandera, L.M. Continuous ID3 algorithm with fuzzy entropy measures. In *Proceedings of the IEEE International Conference on Fuzzy Systems*, 469-476, San Diego, California, March 1992.
- Clark, P. The CN2 induction algorithm. *Machine Learning*, 3(4):261-283, 1989.
- Clark, P. and Boswell, R. Rule induction with CN2: some recent improvements. In *Machine Learning – Proceedings of the Fifth European Conference (EWSL-91)*, 151-163, Yves Kodratoff, Berlin, 1991. Springer-Verlag.

- Clark, P. and Niblett, T. The CN2 induction algorithm. *Machine Learning*, 3:261-283, 1989.
- Cohen, W.W. Fast effective rule induction. In *Proceedings of the Twelfth International Conference on Machine Learning*, 115-123, Lake Tahoe, California, 1995.
- Cost, S. and Salzberg, S. A weighted nearest neighbour algorithm for learning with symbolic features. *Machine Learning*, 10:57-78, 1993.
- Craven, M.W. *Extracting comprehensible models from trained neural networks*. Ph.D. dissertation University of Wisconsin-Madison, 1996.
- Craven, M.W. and Shavlik, J.W. Using sampling and queries to extract rules from trained neural networks. In *Proceedings of the Eleventh International Conference on Machine Learning*, 37-45, New Brunswick, New Jersey, 1994. Morgan Kaufmann.
- Craven, M.W. and Shavlik, J.W. Extracting tree-structured representations of trained networks. In *Advances in Neural Information Processing Systems*, 8:24-30, 1996. MIT Press, Cambridge, Massachusetts, Touretzky, D., Mozer, M. and Hasselmo, M. (eds.).
- Curram, S.P. and Mingers, J. Neural networks, decision tree induction and discriminant analysis: an empirical comparison. *Journal of the Operational Research Society*, 45(4):440-450, 1994.
- Darwin, C. *On the Origin of Species by Means of Natural Selection*, 1859. Murray, London, United Kingdom.
- Datta, B.N. *Numerical Linear Algebra*, 1995. Brookes / Cole Publishing Company, New York.
- Day, R.W. and Quinn, G.P. Comparisons of treatments after an analysis of variance in ecology. *Ecological Monographs*, 59:433-463, 1989.
- DeGroot, M.H. *Probability and Statistics*, 2<sup>nd</sup> Edition, 1986. Addison-Wesley, Reading, Massachusetts.
- Domingos, P. The RISE system: conquering without separating. In *Proceedings of the Sixth IEEE International Conference on Tools with Artificial Intelligence*, 704-707, New Orleans, Los Angeles, 1994. IEEE Computer Society Press.
- Domingos, P. Rule induction and instance-based learning: a unified approach. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, 1226-1232, Montreal, Canada, 1995. Morgan Kaufmann.
- Domingos, P. Two-way induction. *International Journal on Artificial Intelligence Tools*, 5:113-125, 1996a.
- Domingos, P. Using partitioning to speed up specific-to-general rule induction. In *Proceedings of the AAAI-96 Workshop on Integrating Multiple Learned Models*, 29-34, Portland, Oregon, 1996b. AAAI Press.
- Domingos, P. From instances to rules: a comparison of biases. In *Proceedings of the Third International Workshop on Multistrategy Learning*, 147-154, Harpers Ferry, WV, 1996c. AAAI Press.
- Domingos, P. Linear-time rule induction. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, 96-101, Portland, Oregon, 1996d. AAAI Press.
- Domingos, P. Unifying instance-based and rule-based induction. *Machine Learning*, 24:141-168, 1996e.



- Domingos, P. and Pazzani, M. Beyond independence: conditions for the optimality of the simple Bayesian classifier. In *Proceedings of the Thirteenth International Conference on Machine Learning*, 105-112, Bari, Italy, 1996. Morgan Kaufmann.
- Donald, J.H. Rule induction – machine learning techniques. *Computing & Control Engineering Journal*, 5(5):249-255, October 1994.
- Dougherty, J., Kohavi, R. and Sahami, M. Supervised and unsupervised discretization of continuous features. *Machine Learning: Proceedings of the Twelfth International Conference*, 194-202, 1995. Morgan Kaufmann Publishers, Frieditis, A. and Russell, S. (eds.).
- Esposito, F., Malerba, D. and Semeraro, F. A further study of pruning methods in decision tree induction. In *Preliminary Papers of the Fifth International Workshop on Artificial Intelligence and Statistics (AI & Statistics-95)*, 211-218, Fort Lauderdale, Florida, January 1995. Society for AI and Statistics.
- Evans, B. and Fisher, D. Overcoming process delays with decision tree induction. *IEEE Expert*, 60-66, February 1994.
- Fayyad, U.M. and Irani, K.B. On the handling of continuous-valued attributes in decision tree generation. *Machine Learning*, 8:87-102, 1992.
- Fisher, R.A. The use of multiple measurements in taxonomic problems. *Annal of Eugenics*, 7(2):179-188, 1936.
- Flexer, A. Statistical evaluation of neural network experiments: minimum requirements and current practice. In *Proceedings of the Thirteenth European Meeting on Cybernetics and Systems Research (Cybernetics and Systems '96)*, 1005-1008, 1996. Austrian Society for Cybernetic Studies, Vienna, Trappl, R. (ed.).
- Fogler, H.S. *Elements of Chemical Reaction Engineering*, 2<sup>nd</sup> Edition, 1992. Prentice-Hall, Englewood Cliffs, New Jersey.
- Frank, E., Wang, Y., Inglis, S., Holmes, G. and Witten, I.H. Using model trees for classification. *Machine Learning*, 32:63-76, 1998.
- Frey, P.W. and Slate, D.J. Letter recognition using Holland-style adaptive classifiers. *Machine Learning*, 6:161-182, 1991.
- Freedman, D.A. A note on screening regression equations. *The American Statistician*, 37(2):152-155, 1983.
- Friedman, J.H. Multivariate adaptive regression splines. *Annals of Statistics*, 19(1):1-141, 1991.
- Fritzke, B. Growing cell structures – a self-organising network in  $k$  dimensions. In *Artificial Neural Networks*, 2:1051-1056, 1992. North-Holland, Amsterdam.
- Fritzke, B. Fast learning with incremental RBF networks. *Neural Processing Letters*, 1(1):2-5, 1994a.
- Fritzke, B. Growing cell structures – a self-organizing network for unsupervised and supervised learning. *Neural Networks*, 7(9):1441-1460, 1994b.
- Fritzke, B. A growing neural gas network learns topologies. In *Advances in Neural Information Processing Systems*, Volume 7, 625-632, 1995a. MIT Press, Cambridge, Massachusetts, Tesauro, G., Touretzky, D.S. and Leen, T.K. (eds.).

- Fritzke, B. Incremental learning of local linear mappings. In *Proceedings of the International Conference on Artificial Neural Networks (ICANN '95)*, 217-222, Paris, France, 1995b. EC2 and Cie, Fogelman, F. and Gallinari, P. (eds.).
- Fritzke, B. Incremental neuro-fuzzy systems. In *Proceedings of Applications of Soft Computing, SPIE International Symposium on Optical Science, Engineering and Instrumentation*, San Diego, 27 July – 1 August 1997.
- Fu, L. Rule learning by searching on adapted nets. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, 590-595, Anaheim, California, 1991. AAAI/MIT Press.
- Furnival, G.M. and Wilson, R.W. Regression by leaps and bounds. *Technometrics*, **16**(4):499-511, 1974.
- Gallant, S.I. *Neural Network Learning and Expert Systems*, 1993. MIT Press, Cambridge, Massachusetts.
- Garey, M.R. and Johnson, D.S. *Computers and Intractability: a Guide to the Theory of NP-Completeness*, 1979. Freeman and Co., San Francisco, California.
- Gascuel, O., Bouchon-Meunier, B., Caraux, G., Gallinari, P., Guénoche, A., Guermeur, Y., Lechevallier, Y., Marsala, C., Miclet, L., Nicolas, J., Nock, R., Ramdani, M., Sebag, M., Tallur, B., Venturini, G. and Vitte, P. Twelve numerical, symbolic and hybrid supervised classification methods. *International Journal of Pattern Recognition and Artificial Intelligence*, **12**(5):517-571, 1998.
- Gerald, C.F. and Wheatley, P.O. *Applied Numerical Analysis*, 4<sup>th</sup> Edition, 1989. Addison-Wesley, Reading, Massachusetts.
- Glendinning, P. *Stability, Instability and Chaos*. Cambridge University Press, New York, 1994.
- Glover, F. Tabu search – part I. *ORSA Journal on Computing*, **1**(3):190-206, 1989.
- Glover, F. Tabu search – part II. *ORSA Journal on Computing*, **2**(1):4-32, 1990.
- Goldberg, D.E. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, Reading, Massachusetts, 1989.
- Goldschmidt, O., Hochbaum, D.S. and Yu, G. A modified greedy heuristic for the set covering problem with improved worst case bound. *Information Processing Letters*, **48**:305-310, 1993.
- Golub, G.H. and van Loan, C.F. *Matrix Computations*, 2<sup>nd</sup> Edition, 1993. John Hopkins University Press, Baltimore.
- Goodrich, M.T., Mirelli, V., Orletsky, M. and Salowe, J. *Decision tree construction in fixed dimensions: being global is hard but local greed is good*. Technical Report TR-95-1, Department of Computer Science, John Hopkins University, Baltimore, Maryland, May 1995.
- Greene, P.D. and Smith, S.F. Competition-based induction of decision models from examples. *Machine Learning*, **13**(2/3):229-257, 1993.
- Grey, P. and Scott, S.K. Autocatalytic reactions in the isothermal, continuous stirred tank reactor. Isolates and other forms of instability. *Chemical Engineering Science*, **38**:29-43, 1983.
- Grey, P. and Scott, S.K. Autocatalytic reactions in the isothermal, continuous stirred tank reactor. Oscillations and instabilities in the system  $A + 2B \rightarrow 3B$ ;  $B \rightarrow C$ . *Chemical Engineering Science*, **39**:1087-1097, 1984.

- Hamker, F. and Heinke, D. *Implementation and comparison of growing neural gas, growing cell structures and fuzzy ARTMAP*. Technical Report 1/97, Technical University Ilmenau, Ilmenau 1997.
- Harrison, D. and Rubinfeld, D.L. Hedonic prices and the demand for clean air. *Journal of Environmental Economics & Management*, 5:81-102, 1978.
- Hartman, E., Keeler, J.D, and Kowalski, J. Layered neural networks with gaussian hidden units as universal approximators. *Neural Computation*, 2:210-215, 1990.
- Haykin, S. *Neural networks: a comprehensive foundation*, 1994. Macmillan College Publishing Company, New York.
- Heath, D., Kasif, S. and Salzberg, S. Induction of oblique decision trees. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence (IJCAI-93)*, 1002-1007, Chambery, France, 1993. Morgan Kaufmann, San Mateo, California, Bajcsy, R. (ed.).
- Heath, D., Kasif, S. and Salzberg, S. Committees of decision trees. In *Cognitive Technology: In Search of a Humane Interface*, 305-317, 1996. Elsevier Science B.V., Amsterdam, Gorayska, B. and Mey, J. (eds.).
- Hebb, D.O. *The organization of behaviour*, 1949. John Wiley & Sons, New York.
- Heileman, G.L. *Data structures, algorithms, and object-oriented programming*, 1996. McGraw-Hill International, New York.
- Holte, R.C. Very simple classification rules perform well on most commonly used datasets. *Machine Learning*, 11:63-90, 1993.
- Holland, J.H. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, 1975.
- Hong, S.E. R-MINI: an iterative approach for generating minimal rules from examples. *IEEE Transaction on Knowledge and Data Engineering*, 9(5):709-717, 1997.
- Hong, T.-P. and Lee, C.-Y. Induction of fuzzy rules and membership functions from training examples. *Fuzzy Sets and Systems*, 84:33-47, 1996.
- Hwang, Y.-S. and Bang, S.-Y. An efficient method to construct a radial basis function neural network classifier. *Neural Networks*, 10(8):1495-1503, 1997.
- Hyafil, L. and Rivest, R.L. Constructing optimal binary decision trees is NP-complete. *Information Processing Letters*, 5(1):15-17, 1976.
- Ichihashi, H., Shirai, T., Nagasaka, K. and Miyoshi, T. Neuro-fuzzy ID3: a method of inducing fuzzy decision trees with linear programming for maximizing entropy and an algebraic method for incremental learning. *Fuzzy Sets and Systems*, 81:157-167, 1996.
- Ishibuchi, H., Murata, T. and Türksen, I.B. Single-objective and two-objective genetic algorithms for selecting linguistic rules for pattern classification problems. *Fuzzy Sets and Systems*, 89:135-150, 1997.
- Ishibuchi, H., Nozaki, K., Yamamoto, N. and Tanaka, H. Construction of fuzzy classification systems with rectangular fuzzy rules using genetic algorithms. *Fuzzy Sets and Systems*, 65:237-253, 1994.
- Ishibuchi, H., Nozaki, K., Yamamoto, N. and Tanaka, H. Selecting fuzzy if-then rules for classification problems using genetic algorithms. *IEEE Transactions on Fuzzy Systems*, 3(3):260-270, 1995.

- Ittner, A., Zeidler, J., Rossius, R., Dilger, W. and Schlosser, M. Feature space partitioning by non-linear and fuzzy decision trees. In *Proceedings of the Seventh International Fuzzy Systems Association World Congress (IFSA'97)*, 1:394-398, Prague, Czech Republic, June 1997.
- Jäkel, J. Fuzzy model identification based on a genetic algorithm and optimization techniques. In *Proceedings of the Fifth European Congress on Intelligent Techniques and Soft Computing (EUFIT'97)*, 1:774-778, Aachen, Germany, September 1997.
- Jang, J.-S.R. ANFIS: adaptive-network-based fuzzy inference system. *IEEE Transactions on Systems, Man, and Cybernetics*, **23**(3):665-685, May/June 1993.
- Jang, J.-S.R. Structure determination in fuzzy modeling: a fuzzy CART approach. In *Proceedings of the IEEE Conference on Fuzzy Systems*, 480-485, 1994.
- Jang, J.-S.R. and Sun, C.-T. Functional equivalence between radial basis function networks and fuzzy inference systems. *IEEE Transactions on Neural Networks*, **4**(1):156-159, January 1993.
- Jang, J.-S.R., Sun, C.-T. and Mizutani, E. *Neuro-Fuzzy and Soft Computing*, 1997. Prentice-Hall, New Jersey.
- Janikow, C.Z. Fuzzy decision trees: issues and methods. *IEEE Transactions on Systems, Man, and Cybernetics – Part B: Cybernetics*, **28**(1):1-14, February 1998.
- Johansen, T.A. Fuzzy model based control: stability, robustness, and performance issues. *IEEE Transactions on Fuzzy Systems*, **2**(3):221-234, August 1994.
- Johannsen, G. and Alty, J.L. Knowledge engineering for industrial expert systems. *Automatica*, **27**(1):97-114, 1991.
- Johnston, R. Fuzzy logic control. *GEC Journal of Research*, **11**(2):99-109, 1994.
- Joo, Y.H., Hwang, H.S., Kim, K.B. and Woo, K.B. Fuzzy system modeling by fuzzy partition and GA hybrid schemes. *Fuzzy Sets and Systems*, **86**:279-288, 1997.
- Joubert, H.W., Theron, P.L. Personal communication, January 1997.
- Joubert, H.W., Theron, P.L. and Lange, T. The use of neural networks for gas loop process optimisation. *SA Instrumentation and Control*, 44-51, September 1996.
- Juang, C-F. and Lin, C-T. An on-line self-constructing neural fuzzy inference network and its applications. *IEEE Transactions on Fuzzy Systems*, **6**(1):12-32, February 1998.
- Judd, K. and Mees, A. On selecting models for nonlinear time series. *Physica D*, **82**:426-444, 1995.
- Judge, G. Improved prediction in the presence of multicollinearity. *Journal of Econometrics*, **35**:83-100, 1987.
- Kahneman D., Slovic, P. and Tversky, A. *Judgement Under Uncertainty: Heuristics and Biases*, 1982. Cambridge University Press, Cambridge, England.
- Karayiannis, N.B. and Mi, G.W. Growing radial basis neural networks: merging supervised and unsupervised learning with network growth techniques. *IEEE Transactions on Neural Networks*, **8**(6):1492-1506, November 1997.
- Karp, R.M. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, 1972. Plenum, New York, Miller, R.E. and Thatcher, J.W. (eds.).

- Karr, C.L. and Gentry, E.J. Control of pH using genetic algorithms. *IEEE Transactions on Fuzzy Systems*, 1(1):46-53, January 1993.
- Karr, C.L. and Weck, B. Computer modelling of mineral processing equipment using fuzzy mathematics. *Minerals Engineering*, 9(2):183-194, 1996.
- Kasabov, N.K. *Foundations of Neural Networks, Fuzzy Systems, and Knowledge Engineering*, 1996. MIT Press, Cambridge, Massachusetts.
- Kattan, M. Inductive expert systems vs. human experts. *AI Expert*, 32-38, April 1994.
- Kaufmann, S. and Levin, S. Towards a general theory of adaptive walks on rugged landscapes. *Journal of Theoretical Biology*, 128:11-45, 1987.
- Kim, H. and Koehler, G.J. PAC-learning a decision tree with pruning. *European Journal of Operational Research*, 94(2):405-418, 1996.
- Kim, E., Park, Minkee, Ji, S. and Park, Mignon. A new approach to fuzzy modeling. *IEEE Transactions on Fuzzy Systems*, 5(3):328-337, August 1997.
- King, R.D., Feng, C. and Sutherland, A. STATLOG: comparison of classification algorithms on large real-world problems. *Applied Artificial Intelligence*, 9(3):289-333, 1995.
- Kirkpatrick, S., Gelatt, C.D. and Vecchi, M.P. Optimization by simulated annealing. *Science*, 220(4598):671-680, 1983.
- Klir, G.J. and Yuan, B. *Fuzzy Sets and Fuzzy Logic: Theory and Applications*, 1995. Prentice Hall, New Jersey.
- Kobayashi, K., Ogata, H. and Mural, R. A method of inducing fuzzy rules and membership functions. *Control Engineering Practice*, 1(2):283-290, 1993.
- Koch, T. and Fehsenfeld, B. Discovering expert system rules in data sets. *Expert Systems with Applications*, 8(2):287-294, 1995.
- Kohonen, T. Self-organized formation of topologically correct feature maps. *Biological Cybernetics*, 43:59-69, 1982.
- Kohavi, R., John G.H. Wrappers for feature subset selection. *Artificial Intelligence*, 97(1-2):273-324, 1997.
- Kononenko, I., Šimec, E. and Robnik-Šikonja, M. Overcoming the myopia of inductive learning algorithms with ReliefF. *Applied Intelligence*, 7:39-55, 1997.
- Kosko, B. *Neural Networks and Fuzzy Systems*, 1992. Prentice-Hall, Englewood Cliffs, New Jersey.
- Kosko, B. *Fuzzy Engineering*, 1997. Prentice-Hall, New Jersey.
- Lang, K.J. and Witbrock, M.J. Learning to tell two spirals apart. In *Proceedings of the 1988 Connectionist Models Summer School*, 1988. Morgan Kaufmann.
- Langari, R., Wang, L. and Yen, J. Radial basis function networks, regression weights, and the expectation-maximization algorithm. *IEEE Transactions on Systems, Man, and Cybernetics – Part A: Systems and Humans*, 27(5):613-623, September 1997.
- Lawson, C.L. and Hanson, R.J. *Solving Least Squares Problems*. Society for Industrial and Applied Mathematics, Philadelphia, 1995.



- Lee, D.H. and Park, D. An efficient algorithm for fuzzy weighted average. *Fuzzy Sets and Systems*, **87**:39-45, 1997.
- Lin, C.-T. and Lee, C.S.G. Reinforcement structure / parameter learning for neural-network-based fuzzy logic control systems. *IEEE Transactions on Fuzzy Systems*, **2**(1):46-63, February 1994.
- Lin, C.-J. and Lin, C.-T. An ART-based fuzzy adaptive learning control network. *IEEE Transactions on Fuzzy Systems*, **5**(4):477-496, November 1997.
- Linkens, D.A. and Kandiah, S. Long-range predictive control using fuzzy process models. *Transactions IChemE*, **74**(Part A):77-88, January 1996.
- Lippmann, R.P. Pattern classification using neural networks. *IEEE Communications Magazine*, **27**:47-54, 1989.
- Lotfi, A. and Tsoi, A.C. Learning fuzzy inference systems using an adaptive membership function scheme. *IEEE Transactions on Systems, Man, and Cybernetics - Part B: Cybernetics*, **26**(2):326-331, April 1996.
- Lovell, B.C. and Bradley, A.P. The multiscale classifier. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **18**(2):124-137, 1996.
- Lynch, D.T. Chaotic behaviour of reaction systems: parallel cubic autocatalators. *Chemical Engineering Science*, **47**(2):347-355, 1992.
- Mackey, M. and Glass, L. Oscillation and chaos in physiological control systems. *Science*, **197**:287, 1977.
- Mandami, E.H. and Assilian, S. An experiment in linguistic synthesis with a fuzzy logic controller. *International Journal of Man-machine Studies*, **7**(1):1-13, 1975.
- Mangasarian, O.L. and Wolberg, W.H. Cancer diagnosis via linear programming. *SIAM News*, **23**(5):1-18, September 1990.
- Mangasarian, O.L., Setiono, R. and Wolberg, W.H. Pattern recognition via linear programming: theory and application to medical diagnosis. In *Large-Scale Numerical Optimization*, 22-30, 1990. SIAM Publications, Philadelphia, Coleman, T.F. and Li, Y. (eds.).
- Martello, S. and Toth, P. *Knapsack problems: algorithms and computer implementations*. Wiley Interscience, Chichester, United Kingdom, 1990.
- Martinetz, T. Competitive Hebbian learning rule forms perfectly topology preserving maps. In *Proceedings of the International Conference on Artificial Neural Networks (ICANN 93)*, 426-438, Amsterdam, 1993. Springer, New York.
- Martinetz, T. and Schulten, K. A "neural-gas" network learns topologies. In *Artificial Neural Networks*, 397-402, 1991. North-Holland, Amsterdam, Kohonen, T., Mäkisara, K., Simula, O. and Kangas, J. (eds.).
- Martinetz, T. and Schulten, K. Topology representing networks. *Neural Networks*, **7**(3):507-522, 1994.
- Michalski, R.S. Variable-valued logic and its applications to pattern recognition and machine learning. In *Computer Science and Multiple-valued Logic: Theory and Applications*, 506-534, 1975. North Holland, Rine, D.C. (ed.).

- Michalski, R.S. A theory and methodology of inductive learning. In *Machine Learning: An Artificial Intelligence Approach*, 1:83-134, 1983. Morgan Kaufmann, Los Altos, California, Michalski, R.S., Carbonell, J.G. and Mitchell, T.M. (eds.).
- Michalski, R.S., Mozetic, I., Hong, J. and Lavrac, N. The multi-purpose incremental learning system AQ15 and its testing application to three medical domains. In *Proceedings of the Fifth National Conference on Artificial Intelligence (AAAI-86)*, 1041-1045, Philadelphia, 1986. AAAI Press.
- Michie, D. Directions in machine intelligence. *Computer Bulletin*, 9-11, September / October 1992.
- Miller, A.J. *Subset Selection in Regression*. Chapman and Hall, 1990.
- Mingers, J. Expert systems - rule induction with statistical data. *Journal of the Operational Research Society*, **38**(1):39-47, 1987.
- Mingers, J. An empirical comparison of pruning methods for decision tree induction. *Machine Learning*, **4**:227-243, 1989.
- Moody, J.E. Note on generalization, regularization, and architecture selection in nonlinear learning systems. In *Neural Networks for Signal Processing*, 1-10, 1991. IEEE Press, Piscataway, New Jersey, Juang, B.H., Kung, S.Y. and Kamm, C.A. (eds.).
- Moody, J.E. The effective number of parameters: an analysis of generalization and regularization in nonlinear learning systems. In *Advances in Neural Information Processing Systems*, **4**:847-854, 1992. Morgan Kaufmann Publishers, San Mateo, California, Moody, J.E., Hanson, S.J. and Lippmann, R.P. (eds.).
- Moody, J. and Darken, C. Fast learning in networks of locally-tuned processing units. *Neural Computation*, **1**:281-294, 1989.
- Moolman, D.W., Aldrich, C., Van Deventer, J.S.J. and Bradshaw, D.J. The interpretation of flotation froth surfaces by using digital image analysis and neural networks. *Chemical Engineering Science*, **50**(22):3501-3513, 1995.
- Muggleton, S. *Inductive Acquisition of Expert Knowledge*, 1990. Turing Institute Press, Glasgow in association with Addison-Wesley Publishing Company, New York.
- Murphy, P.M. and Pazzani, M.J. ID2-of-3: Constructive induction of M-of-N concepts for discriminators in decision trees. In *Proceedings of the Eighth International Machine Learning Workshop*, 183-187, Evanston, Illinois, 1991. Morgan Kaufmann, San Mateo, California.
- Murphy, P.M. and Pazzani, M.J. Exploring the decision forest: an empirical investigation of Occam's razor in decision tree induction. *Journal of Artificial Intelligence Research*, **1**:257-275, 1994.
- Murthy, S. *On Growing Better Decision Trees from Data*. Ph.D. Dissertation, John Hopkins University, Baltimore, Maryland, 1996.
- Murthy, S. and Salzberg, S. Lookahead and pathology in decision tree induction. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI-95)*, 1025-1031, Montreal, Canada, August 1995. Morgan Kaufmann, San Mateo, California, Mellish, C. (ed.).
- Naumov, G.E. NP-completeness of problems of construction of optimal decision trees. *Soviet Physics: Doklady*, **36**(4):270-271, April 1991.
- Ng, K. and Lippmann, R.P. Practical characteristics of neural network and conventional pattern classifiers. In *Advances in Neural Information Processing Systems*, Volume 3, 970-976, 1991. San Mateo, California. Lippmann, et al. (eds.).

- Nie, J. Constructing fuzzy model by self-organizing counterpropagation network. *IEEE Transactions on Systems, Man and Cybernetics*, **25**(6):963-970, 1995.
- Nie, J.H. and Lee, T.H. Rule-based modeling: fast construction and optimal manipulation. *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, **26**(6):728-738, November 1996.
- Nie, J.H. and Lee, T.H. Self-organizing rule-based control of multivariable nonlinear servomechanisms. *Fuzzy Sets and Systems*, **91**:285-304, 1997.
- Nieuwoudt, I. Personal communication, October 1998.
- Nonobe, K. and Ibaraki, T. A tabu search approach to the constraint satisfaction problem as a general problem solver. *European Journal of Operational Research*, **106**(2-3):599-623, 1998.
- Norton, S.W. Generating better decision trees. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence (IJCAI-89)*, 800-805, San Mateo, California, 1989. Sridharan, N.S. (ed.).
- Nozaki, K., Ishibuchi, H. and Tanaka, H. Adaptive fuzzy rule-based classification systems. *IEEE Transactions on Fuzzy Systems*, **4**(3):238-250, August 1996.
- Oliver, J.J. and Hand, D. *Introduction to minimum encoding inference*. Technical Report 4-94, Department of Statistics, Open University, Walton Hall, United Kingdom, 1994. (Also published as Technical Report 205, Department of Computer Science, Monash University, Clayton, Victoria, Australia, 1994.)
- Omohundro, S.M. *The Delaunay triangulation and function learning*. Technical Report TR-90-0001, International Computer Science Institute, Berkeley, 1990.
- Pagallo, G. and Haussler, D. Boolean feature discovery in empirical learning. *Machine Learning*, **5**:71-99, 1990.
- Papadimitriou, C.H. and Steiglitz, K. *Combinatorial Optimization: Algorithms and Complexity*. Prentice-Hall, Englewood Cliffs, New Jersey, 1982.
- Park, D., Kandel, A. and Langholz, G. Genetic-based new fuzzy reasoning models with application to fuzzy control. *IEEE Transactions on Systems, Man and Cybernetics*, **24**:39-47, 1994.
- Park, J. and Sandberg, I.W. Universal approximation using radial basis function networks. *Neural Computation*, **3**:246-257, 1991.
- Pham, D.T. An introduction to artificial neural networks. In *Neural Networks for Chemical Engineers*, Chapter 1, 1-19, 1995. Elsevier Science, Amsterdam, Bulsari, A.B. (ed.).
- Pham, D.T. and Dimov, M.S. An efficient algorithm for automatic knowledge acquisition. *Pattern Recognition*, **30**(7):1137-1144, July 1997.
- Platt, J. A resource allocating network for function interpolation. *Neural Computation*, **3**:213-225, 1991.
- Poggio, T. and Girosi, F. Networks for approximation and learning. In *Proceedings of the IEEE*, **78**(9):1485-1497, September 1990.
- Pokorný, M., Čermák, P., Ošmera, P. and Šimoník, I. Fuzzy models identification and tuning using the genetic algorithms and neural networks. In *Proceedings of the Fifth European Congress on*

- Intelligent Techniques and Soft Computing (EUFIT'97)*, 1:302-306, Aachen, Germany, September 1997.
- Prechelt, L. A quantitative study of experimental evaluations of neural network algorithms: current research and practice. *Neural Networks*, 9:457-462, 1996.
- Quinlan, J.R. Discovering rules from large collections of examples: a case study. In *Expert Systems in the Micro Electronic Age*, 1979. Edinburgh University Press, Edinburgh, Michie, D. (ed.).
- Quinlan, J.R. Learning efficient classification procedures and their application to chess end games. In *Machine Learning: An Artificial Intelligence Approach*, 1:463-482, 1983. Morgan Kaufmann, Los Altos, California, Michalski, R.S., Carbonell, J.G. and Mitchell, T.M. (eds.).
- Quinlan, J.R. Induction of decision trees. *Machine Learning*, 1:81-106, 1986.
- Quinlan, J.R. Simplifying decision trees. *International Journal of Man-Machine Studies*, 27:221-234, 1987.
- Quinlan, J.R. Decision trees and multi-valued attributes. *Machine Intelligence*, 11:305-318, 1988.
- Quinlan, J. R. Learning with continuous classes. In *Proceedings of the 5th Australian Joint Conference on Artificial Intelligence*, 343-348, 1992. World Scientific, Adams, N. and Sterling, L. (eds.).
- Quinlan, J.R. *C4.5: Programs for machine learning*, 1993a. Morgan Kaufmann, San Mateo, California.
- Quinlan, J.R. Combining instance-based and model-based learning. In *Proceedings of the Tenth International Conference of Machine Learning*, 236-243, University of Massachusetts, Amherst. June 27-29, 1993b. Morgan Kaufmann, California.
- Quinlan, J.R. Bagging, boosting, and C4.5. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI-96)*, 1996. AAAI Press, Menlo Park, California.
- Quinlan, J.R. and Cameron-Jones, R.M. Oversearching and layered search in empirical learning. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI '95)*, 1019-1024, Montreal, 1995. Morgan Kaufmann.
- Quinlan, J.R. and Rivest, R.L. Inferring decision trees using the minimum description length principle. *Information and Computation*, 80:227-248, 1989.
- Ragavan, H. and Rendell, L. Lookahead feature construction for learning hard concepts. In *Proceedings of the Tenth International Conference on Machine Learning (ML-93)*, 252-259, University of Massachusetts, Amherst, Massachusetts, June 1993. Morgan Kaufmann, Utgoff, P.E. (ed.).
- Reeves, C.R. (ed.) *Modern Heuristic Techniques for Combinatorial Problems*, 1993. Blackwell Scientific Publications, Osney Mead, Oxford.
- Reimann, J., John, H. and Seeger, W. Transition from slug to annular flow in horizontal pipes. *Multiphase Science and Technology*, Volume 6, 1992. Hemisphere Publishing Corporation, New York, Hewitt, G.F., Delhay, J.M., Zuber, N. (eds.).
- Rendell, L.A. and Seshu, R. Learning hard concepts through constructive induction: framework and rationale. *Computational Intelligence*, 6(4):247-270, 1990.
- Riddle, P., Segal, R. and Etzioni, O. Representation design and brute-force induction in a Boeing manufacturing domain. *Applied Artificial Intelligence*, 8:125-147, 1994.
- Rissanen, J. A universal prior for integers and estimation by minimum description length principle. *Annals of Statistics*, 11:416-431, 1983.

- Rumelhart, D.E., Hinton, G.E. and Williams, R.J. Learning representations by back-propagating errors. In *Neurocomputing: Foundations of Research*, 696-699, 1986. MIT Press, Cambridge, Massachusetts.
- Russo, M. FuGeNeSys – a fuzzy genetic neural system for fuzzy modeling. *IEEE Transactions on Fuzzy Systems*, 6(3):373-388, 1998.
- Sachs, L. *Applied Statistics: A Handbook of Techniques*, 2<sup>nd</sup> Edition, 1984. Springer-Verlag, New York.
- Saito, K. and Nakano, R. Medical diagnostic expert system based on PDP model. In *Proceedings of the IEEE International Conference on Neural Networks*, 255-262, San Diego, California, 1988. IEEE Press.
- Salzberg, S. On comparing classifiers: pitfalls to avoid and a recommended approach. *Data Mining and Knowledge Discovery*, 1:317-327, 1997.
- Santagati, N.A., Lo Pinto, E. and Russo, M. Fuzzy logic can help medicinal chemistry. In *Proceedings of the Artificial Neural Networks in Engineering Conference*, 297-302, St. Louis, June 1995.
- Saraiva, P.M. and Stephanopoulos, G. Continuous process improvement through inductive and analogical learning. *AIChE Journal*, 38(2):161-183, February 1992.
- Schmitz, G.P.J. *Combinatorial Evolution of Regression Nodes*. Ph.D. Dissertation, Department of Chemical Engineering, University of Stellenbosch, December 1999.
- Schmitz, G., Aldrich, C. and Gouws, F.S. ANN-DT: an algorithm for extraction of decision trees from artificial neural networks. Accepted for publication in *IEEE Transaction on Neural Networks*, 1999.
- Schultz, M.H. *Theory and applications of spline functions*, 1969. New York Academic Press, New York, Greville, T.N.E. (ed.); Publication number 22 of the Mathematics Research Center, University of Wisconsin.
- Seber, G.A.F. *Linear Regression Analysis*, 1977. Wiley, New York.
- Seisto, S. and Dillon, T. Automated knowledge acquisition of rules with continuously valued attributes. In *Proceedings of the Twelfth International Conference on Expert Systems and their Applications*, 645-656, Avignon, France, May 1992.
- Seshu, R. Solving the parity problem. In *Proceedings of the Fourth European Working Session on Learning*, 263-271, Montpellier, France, 1989. Morgan Kaufmann.
- Setnes, M. *Fuzzy rule-based simplification using similarity measures*. M.Sc. Thesis, Department of Electrical Engineering, Delft University of Technology, Delft, 1995.
- Setnes, M., Babuška, R., Kaymak, U. and van Nauta Lemke, H.R. Similarity measures in fuzzy rule simplification. *IEEE Transactions on Systems, Man, and Cybernetics – Part B: Cybernetics*, 28(3):376-386, 1998.
- Shavlik, J.W., Mooney, R.J. and Towell, G.G. Symbolic and neural learning algorithms: an experimental comparison. *Machine Learning*, 6(2):111-143, 1991.
- Sigillito, V. G., Wing, S. P., Hutton, L. V. and Baker, K. B. Classification of radar returns from the ionosphere using neural networks. *Johns Hopkins APL Technical Digest*, 10:262-266, 1989.



- Simpson, P.K. Fuzzy min-max neural networks - part 1: classification. *IEEE Transactions on Neural Networks*, 3(5):776-786, September 1992.
- Simpson, P.K. Fuzzy min-max neural networks – part 2: clustering. *IEEE Transactions on Fuzzy Systems*, 1(1):32-45, February 1993.
- Sinclair, M. Comparison of the performance of modern heuristics for combinatorial optimization on real data. *Computers and Operations Research*, 20(7):687-695, 1993.
- Smets, H.M.G. and Bogaerts, W.F.L. SCC susceptibility analysis of stainless steels in nuclear reactor water: a neural network and expert system approach. *Fuzzy Sets and Systems*, 74:153-162, 1995.
- Smith, J.W., Everhart, J.E., Dickson, W.C., Knowler, W.C. and Johannes, R.S. Using the ADAP learning algorithm to forecast the onset of diabetes mellitus. In *Proceedings of the Symposium on Computer Applications and Medical Care*, 261-265, 1988..IEEE Computer Society Press.
- Snedecor, G.W. and Cochran, W.G. *Statistical methods*, 6<sup>th</sup> Edition, 1967. Iowa State University Press.
- Stone, G.O. An analysis of the delta rule and the learning of statistical associations. In *Parallel Distributed Processing*, 1:444-459, 1986. MIT Press, Cambridge, McClelland, R.D.E. and McClelland, J.L. (eds.).
- Suárez, A. and Lutsko, J.F. Fuzzy decision trees. *Poster Presentation at LEARNING '98*, Madrid, Spain, September 1998.
- Sugeno, M. and Kang, G.T. Structure identification of a fuzzy model. *Fuzzy Sets and Systems*, 28:15-33, 1988.
- Sugeno, M. and Tanaka, K. Successive identification of a fuzzy model and its application to prediction of a complex system. *Fuzzy Sets and Systems*, 42:315-334, 1991.
- Sugeno, M. and Yasukawa, T. A fuzzy-logic-based approach to qualitative modeling. *IEEE Transactions on Fuzzy Systems*, 1(1):7-31, February 1993.
- Sun, C.-T. Rule-base structure identification in an adaptive-network-based fuzzy inference system. *IEEE Transactions on Fuzzy Systems*, 2(1):64-73, February 1994.
- Szczepanik, W., Arciszewski, T. and Wnek, J. Empirical performance comparison of two symbolic learning systems based on selective and constructive induction. In *Proceedings of the IJCAI- 95 Workshop on Machine Learning in Engineering*, Montreal, Canada, August, 1995.
- Takagi, T. and Sugeno, M. Fuzzy identification and its application to modeling and control. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-15(1):116-132, February 1985.
- Tanaka, K. and Sugeno, M. Stability analysis and design of fuzzy control systems. *Fuzzy Sets and Systems*, 45:135-156, 1992.
- Tanaka, M., Ye, J. and Tanino, T. Fuzzy modelling by genetic algorithm with tree-structured individuals. *International Journal of Systems Science*, 27(2):261-268, 1996.
- Tani, T. and Sakoda, M. Fuzzy modeling by ID3 algorithm and its application to prediction of heater outlet temperature. In *Proceedings of the IEEE International Conference on Fuzzy Systems*, 923-930, San Diego, California, March 1992.
- Theron, H. *Specialization by exclusion: an approach to concept learning*. Ph.D. dissertation, Department of Computer Science, University of Stellenbosch, Stellenbosch, South Africa, March 1994.

- Theron, H. and Cloete, I. BEXA: a covering algorithm for learning propositional concept descriptions. *Machine Learning*, 24:5-40, 1996.
- Thrun, S. Extracting rules from artificial neural networks with distributed representations. In *Advances in Neural Information Processing Systems*, 7, 1995. MIT Press, Cambridge, Massachusetts, Tesauro, G., Touretzky, D. and Leen, T. (eds.).
- Tickle, A.B., Orlowski, M. and Diederich, J. DEDEC: decision detection by rule extraction from neural networks, *QUT NRC*, December 1994.
- Ting, K.M. The problem of small disjuncts: its remedy in decision trees. In *Proceedings of the Tenth Canadian Conference on Artificial Intelligence*, 91-97, 1994.
- Tong, R.M. Synthesis of fuzzy models for industrial processes - some recent results. *International Journal of General Systems*, 4:143-162, 1978.
- Tong, R.M. The evaluation of fuzzy models derived from experimental data. *Fuzzy Sets and Systems*, 4:1-12, 1980.
- Torgo, L. Kernel regression trees. In *Proceedings of the poster papers of the European Conference on Machine Learning*. University of Economics, Faculty of Informatics and Statistics, Prague, 1997.
- Towell, G. and Shavlik, J. Extracting refined rules from knowledge-based neural networks. *Machine Learning*, 13(1):71-101, 1993.
- Tsukamoto, Y. An approach to fuzzy reasoning method. In *Advances in fuzzy set theory and applications*, 137-149, 1979. North-Holland, Amsterdam, Gupta, M.M., Ragade, R.K. and Yager, R.R. (eds.).
- Utgoff, P.E. Incremental induction of decision trees. *Machine Learning*, 4:161-186, 1989.
- Utgoff, P.E. and Brodley, C.E. An incremental method for finding multivariate splits for decision trees. In *Proceedings of the Seventh International Conference on Machine Learning*, 58-65, Austin, Texas, 1990. Morgan Kaufmann.
- Vaidyanathan, R. and Venkatasubramanian, V. Representing and diagnosing dynamic process data using neural networks. *Engineering Applications of Artificial Intelligence*, 5(1):11-21, 1992.
- Wallace, C.S. and Patrick J.D. Coding decision trees. *Machine Learning*, 11:7-22, 1993.
- Wang, L. and Langari, R. Building Sugeno-type models using fuzzy discretization and orthogonal parameter estimation techniques. *IEEE Transactions on Fuzzy Systems*, 3(4):454-458, November 1995.
- Wang, L. and Langari, R. Complex systems modeling via fuzzy logic. *IEEE Transactions on Systems, Man, and Cybernetics*, 26(1):100-106, February 1996a.
- Wang, L. and Langari, R. Sugeno model, fuzzy discretization, and the EM algorithm. *Fuzzy Sets and Systems*, 82:279-288, 1996b.
- Wang, L.-X. and Mendel, J.M. Generating fuzzy rules by learning from examples. *IEEE Transactions on Systems, Man, and Cybernetics*, 22(6):1414-1427, November / December 1992.
- Wang, X.Z., Chen, B.H., Yang, S.H. and McGreavy, C. Fuzzy rule generation from data for process operational decision support. *Computers & Chemical Engineering*, 21(Supplement):661-666, 1997.

- Wang, Y. and Witten, I.H. Induction of model trees for predicting continuous classes. In *Proceedings of the poster papers of the European Conference on Machine Learning*. University of Economics, Faculty of Informatics and Statistics, Prague, 1997.
- Waugh, S. *Extending and benchmarking cascade-correlation*. Ph.D. Dissertation, Computer Science Department, University of Tasmania, 1995.
- Weber, R. Automatic knowledge acquisition for fuzzy control applications. In *Proceedings of the International Symposium on Fuzzy Systems*, 9-12, Iizuka, Japan, July 1992a.
- Weber, R. Fuzzy-ID3: a class of methods for automatic knowledge acquisition. In *Proceedings of the International Symposium on Fuzzy Logic & Neural Networks*, 265-268, Iizuka, Japan, July 1992b.
- Weiss, S.M., Galen, R.S. and Tadepalli, P.V. Maximizing the predictive value of production rules. *Artificial Intelligence*, **45**(1/2):47-71, 1990.
- Weiss, S. and Indurkha, N. Optimized rule induction. *IEEE Expert*, **8**(6):61-69, 1993.
- Widrow, B. and Hoff, M.E. Adaptive switching circuits. In *IRE WESCON Convention Record*, 96-104, New York, 1960.
- Wong, C.-C. and Lin, N.-S. Rule extraction for fuzzy modeling. *Fuzzy Sets and Systems*, **88**:23-30, 1997.
- Woodruff, D.L. and Zemel, E. Hashing vectors for tabu search. *Annals of Operations Research*, **41**:123-137, 1993.
- Ying, H. Practical design of nonlinear fuzzy controllers with stability analysis for regulating processes with unknown mathematical models. *Automatica*, **30**(7):1185-1195, 1994.
- Yingwei, L., Sundararajan, N. and Saratchandran, P. A sequential learning scheme for function approximation using minimal radial basis function neural networks. *Neural Computation*, **9**:461-478, 1997.
- Yu, Y. Triangular norms and TNF-sigma algebras. *Fuzzy Sets and Systems*, **16**(3):251-264, 1985.
- Yuan, Y. and Shaw, M.J. Induction of fuzzy decision trees. *Fuzzy Sets and Systems*, **69**:125-139, 1995.
- Yuan, Y. and Zhuang, H. A genetic algorithm for generating fuzzy classification rules. *Fuzzy Sets and Systems*, **84**:1-19, 1996.
- Zadeh, L.A. Outline of a new approach to the analysis of complex systems and decision processes. *IEEE Transactions on Systems, Man, and Cybernetics*, **SMC-3**(1):28-44, January 1973.
- Zeidler, J. and Schlosser, M. Fuzzy handling of continuous-valued attributes in decision trees. In *Proceedings of the ECML-95 Mlnet Familiarization Workshop "Statistics, Machine Learning and Knowledge Discovery in Databases"*, 41-46, Heraklion, Crete, Greece, April 1995.
- Zeidler, J. and Schlosser, M. Continuous-valued attributes in fuzzy decision trees. In *IMPU'96: Proceedings of Information Processing and Management of Uncertainty in Knowledge-based Systems*, 395-400, Granada, Spain, July 1996.
- Zheng, Z. Constructing nominal X-of-N attributes. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, 1064-1070, Montreal, 1995. Morgan Kaufmann.
- Zurada, J.M. *Introduction to artificial neural systems*, 1992. West Publishers.

# Appendix A

## Default Algorithm Parameter Settings

This appendix lists the parameter settings for the algorithms that were evaluated in chapters 6 and 7. Parameter settings are given for CART, BEXA, the Growing Neural Gas algorithm, the CORA algorithm, the radial basis function network, the two multilayer perceptron variants and the MARS algorithm. In addition, a list is given of the different results that were recorded during each experiment for a given algorithm.

### A.1 The CART Decision Tree Algorithm

Different split criteria are used by CART for classification and regression problems. For classification problems the Gini impurity criterion (Breiman, et al., 1984) is used. For regression problems the splitting criterion minimises the least squares error when choosing the best split. During tree induction, both for classification and regression problems, all surrogates count equally and a maximum of five surrogates may be used. Refer to Breiman, et al. (1984) for more details of these concepts.

The information stored for each run of the CART algorithm is the training and validation accuracy of the induced decision tree. If applicable, the testing accuracy is also recorded. Furthermore, the number of tree leaves and the total number of tree nodes (leaf and decision nodes) are also recorded.

The CART software implementation allows one to set the random seed of the algorithm. As far as could be ascertained, different seed values had no effect on the final output of the algorithm. Therefore the algorithm was run once only for each problem.

## A.2 Crisp Rule Induction Using BEXA

Apart from the CART software, none of the software implementations of the algorithms that are evaluated in chapters 6 and 7 are able to use a validation data set in a similar, automatic fashion to the way the CART software can. Therefore all these algorithms, excluding the CORA algorithm, were trained and evaluated several times in order to find the best values for the problem-specific parameters. For the BEXA algorithm the only parameter that was tuned was `beam_width`. The best results obtained using beam widths of either 1 or 10 are reported. Table A.1 below gives a complete list of all the parameter settings that were used. The reader is referred to section 2.2.2.2 and Theron (1994) for more details regarding the meaning of these settings or parameters.

Setting or Parameter	Value(s)
<code>type_of_rules</code>	unordered
<code>display_negated_sets</code>	yes
<code>partitioning_method</code>	one_cut_point
<code>specialize_method</code>	remove_single_values
<code>beam_width</code>	best of 1 or 10
<code>remove_duplicates</code>	yes
<code>prevent_empty_conjunctions</code>	yes
<code>uncover_new_negatives</code>	yes
<code>generate_irredundant_covers</code>	yes
LEF	(laplace,max,0.0000)
<code>significance_test</code>	none
<code>significance_threshold</code>	0
<code>stop_growth_test</code>	log_likelihood_ratio_statistic
<code>stop_growth_threshold</code>	-90
<code>post_pruning_method</code>	prune_like_quinlan
<code>classification_method</code>	classify_with_conjuncts

**Table A.1 BEXA Parameter Values and Settings**

The information recorded for each BEXA run is the training and validation accuracies (or if applicable, the testing accuracy) as well as the number of concepts for both the unpruned and pruned rule models.

As far as the author is aware, BEXA is a deterministic algorithm. Therefore, after the optimal parameter settings were found the algorithm was run once with these optimal settings and the results recorded.



### A.3 The Growing Neural Gas (GNG) Algorithm

The parameter settings used for the GNG algorithm are listed in Table A.2. The second parameter in the table, viz. `switch_to_supervised`, indicates how many cell nodes need to be inserted into the hidden layer of the GNG network before network training switches from unsupervised to supervised mode.

Setting or Parameter	Value(s)
<code>number_of_epochs</code> <sup>1</sup>	50 to 800
<code>switch_to_supervised</code>	after ¼ of nodes inserted
<code>maximum_number_cell_nodes</code>	2 to 150
<code>cell_node_insertion_interval</code> ( $\lambda$ )	1000 iterations <sup>2</sup>
<code>maximum_permitted_connection_age</code> ( $a_{max}$ )	500
<code>winner_reference_vector_update_factor</code> ( $\epsilon_w$ )	0.005
<code>runner_up_reference_vector_update_factor</code> ( $\epsilon_n$ )	0.0001
<code>winner_and_runner_up_resource_reduction_factor</code> ( $\alpha$ )	0.1
<code>cell_node_resource_reduction_factor</code> ( $\beta$ )	0.005
<code>width_update_factor</code> ( $\tau$ )	1.0
<code>delta_learning_method_learning_rate</code>	0.0001
<code>random_seed</code>	5, 7 or 9
<code>width_update_interval</code>	5 epochs

**Table A.2 GNG Parameter Values and Settings**

The results recorded for the GNG algorithm are the training and validation accuracies (or testing accuracy if applicable), and the epoch at which the best training accuracy was obtained. In addition, the number of radial basis functions, concepts and parameters required to completely describe the set of fuzzy rules are also recorded. Finally, the input space overlap of the set of derived fuzzy rules is also stored.

### A.4 The Combinatorial Rule Assembler (CORA)

Table A.3 lists the parameter values that were used by the CORA algorithm in this dissertation. The number of Reactive Tabu Search (RTS) combinatorial search iterations (the `number_of_iterations` parameter) was adjusted according to two factors. The primary factor was the number of iterations required by the algorithm to obtain an acceptable training accuracy. The parameter `number_of_iterations` was assigned at least a value that was roughly twice this number

<sup>1</sup> One epoch equals one presentation of the entire training data set to the network.

<sup>2</sup> An iteration equals the presentation of one training exemplar to the network.

of iterations. The secondary factor was the time that the CORA algorithm took to complete a run. Problems that caused the algorithm to complete iterations slowly were given fewer search iterations to complete.

Setting or Parameter	Value(s)
number_of_iterations	500 to 5000
number_of_rbfs	determined by GNG algorithm <sup>3</sup>
number_of_rules	5 to 30 <sup>4</sup>
maximum_cycles	50
maximum_repetitions	3
chaos_threshold	3
hash_table_size	103591
consequent_magnitude_penalty_factor	0 to 0.1
overlap_penalty_factor	0 to 0.5
minimum_swap_move_threshold	0.01, 0.5 or 0.9
maximum_swap_move_threshold	0.99
swap_move_threshold_decrement	0.0025
rule_reduction_threshold	0.0025
random_seed	5, 7 or 9

**Table A.3 CORA Algorithm Parameter Values**

The maximum\_cycles, maximum\_repetitions and chaos\_threshold parameter values used in this dissertation are identical to those used by Battiti and Tecchiolli (1994a). The hash table size was chosen to be a prime number large enough so that the hash table never filled for any problem considered in this dissertation. Refer to Heileman (1996) for more details regarding hash tables.

When consequent magnitude penalisation was employed the magnitude of the penalisation factor was chosen based on a few shortened, preliminary runs with the given set of training data. For each successive run the magnitude of the penalisation factor was increased until the training accuracy of the generated set of fuzzy rules could not exceed an acceptable level (in comparison to results obtained with no consequent magnitude penalisation). The final penalisation factor value chosen was at most this magnitude.

When overlap penalisation was employed the magnitude of the overlap\_penalty\_factor was principally determined by the magnitude of the overlap of the fuzzy rules derived by the GNG

<sup>3</sup> All of the radial basis functions in the hidden layer of the GNG-trained radial basis function network were given to the RTS algorithm for its combinatorial search.

<sup>4</sup> In the current software implementation, the maximum number of rules assembled by the RTS component of the CORA algorithm must be less than the number of radial basis functions obtained from the GNG-trained radial basis function network.

algorithm. If the latter was large in comparison to the unpenalised fitness of the initial set of fuzzy rules of the RTS component of the CORA algorithm, then the penalty factor was chosen to be relatively small and vice versa. Small penalty factors were in the region of 0.1 and large penalty factors had a maximum magnitude of 0.5. The reason why such a method was used to determine the overlap penalty factor was to ensure that the RTS component spent most effort trying to minimise the regression error of the set of fuzzy rules, rather than the input space overlap of the set of rules.

The value for the `swap_move_threshold_decrement` parameter was chosen so that the RTS component of the CORA algorithm tends to use its search attractor detection and avoidance functionality before significantly increasing the percentage of swaps or moves that are evaluated for a given search iteration.

The principal results that are recorded for each component<sup>5</sup> of the CORA algorithm are as follows. They are the training and validation (or testing, if applicable) accuracies, the number of rules, concepts and parameters used by the trained rule model, and finally the input space overlap of the rule model. Lastly, the RTS iteration at which the best training accuracy was obtained is also recorded.

## A.5 The $k$ -means Radial Basis Function Network

Setting or Parameter	Value(s)
number_of_iterations	10 to 90
number_of_rbf_nodes	5 to 30
rbf_width_factor	1 to 8
random_seed	5, 7 or 9

**Table A.4**  $k$ -means Radial Basis Function Network Settings

Table A.4 gives the parameters that could be set for the  $k$ -means radial basis function network used in this dissertation. The righthand column lists the range of parameter values that were utilised. The number of training iterations was set based on the number of hidden layer nodes that were used. More iterations were used if more radial basis functions were employed. The width of each radial basis function is defined as the distance between the centre of the current radial basis function (for which a width must be determined) and the radial basis function that is

---

<sup>5</sup> The components that are being referred to are the Reactive Tabu Search (RTS) component, the membership function merging (MRG) component and the rule reduction (RED) component.

closest to it in the input space, times the `rbf_width_factor`. An Euclidean norm is used to calculate the distance.

The results stored for each experiment performed with the radial basis function network are the training and validation (or testing, if applicable) accuracies, the number of hidden layer radial basis function nodes that are used and finally the number of training iterations.

## A.6 The Multilayer Perceptron

Two different multilayer perceptron networks were evaluated in this dissertation. Both use the same range of settings. These are listed in Table A.5. The information stored in experiments using either of these two algorithms is the training and validation (or testing, if applicable) accuracies, the number of hidden layer nodes and finally the number of training iterations.

Setting or Parameter	Value(s)
number of epochs	10000 to 100000
number of hidden layer nodes	2 to 20
random seed	5, 7 or 9

**Table A.5 Multilayer Perceptron Training Parameter Settings**

## A.7 Multivariate Adaptive Regression Splines

The settings used by the Multivariate Adaptive Regression Splines (MARS) algorithm for all the problems that were evaluated in this dissertation are listed in Table A.6. Refer to Friedman (1991) for more detailed description of the algorithmic parameters. `maximum_basis_functions` refers to the maximum number of basis functions that can be used to build a model. The version of MARS used in this dissertation utilises the bagging procedure of Breiman (1996). `number_bootstrap_samples` refers to the number of bootstrap repetitions that are used for the bagging procedure.

Setting or Parameter	Default Value(s)
maximum basis functions	15
number bootstrap samples	50
max attributes per basis function	8
random seed	5, 7 or 9

**Table A.6 MARS Training Parameter Settings**

The information stored for each MARS experiment is the validation (or testing, if applicable) accuracy and the so-called “effective number of model parameters”.

# Appendix B

## Additional Figures for the Empirical Evaluation Presented in Chapter 6

This appendix contains all the figures pertaining to the experiments performed for the empirical evaluation of the CORA algorithm presented in chapter 6, but that were not placed in that chapter. The figures can be grouped into the following sets:

- Figures B.1 through B.6 give additional results for section 6.2.1. The figures show how the number of CORA model parameters changes as the number of rules assembled by the CORA algorithm increases.

Each bar of the ten bars in each figure represents the average result over the three different random seed runs performed using either the GNG or the CORA algorithm. The black error line overlaid over each bar represents one standard deviation above and below the average result.

In each figure the first bar represents the results obtained by the GNG algorithm. The number in parentheses following each horizontal axis label is the number of fuzzy rules used by the given algorithm to generate the results presented in the figure. The following nine bars are all results obtained with the CORA algorithm. These nine bars are grouped in three groups of three bars each. For a particular group the first bar represents the results obtained directly after the RTS component of the CORA algorithm has completed execution. The second bar represents the results obtained after both the RTS and MRG components have been executed. Finally, the third bar



represents the results after all three CORA components, viz. the RTS, MRG and RED components, have been employed.

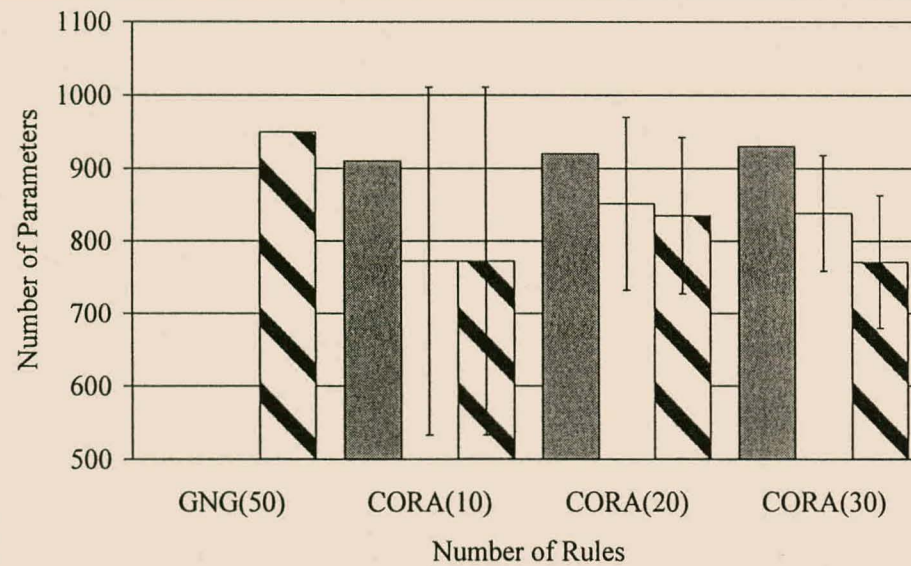
- Figures B.7 through B.24 give additional results for section 6.2.1. In contrast to figures 6.29 through 6.40 and B.1 through B.6, these figures present results of experiments that used a `minimum_swap_move_threshold` of either 0.5 or 0.01. The `minimum_swap_move_threshold` value that is used is indicated in parentheses after each figure caption.
- Figures B.25 through B.28 give parameter complexity results for section 6.2.2 where the effect of the RTS search resolution is studied. In these figures the same number (the maximum) of fuzzy rules are used as are used in figures 6.41 through 6.48 in section 6.2.2. The information in parentheses at the end of each figure caption gives the problem for which the results are applicable as well as how many rules were used by the CORA algorithm to generate the results.

Figures B.29 through B.46 present further results of the search resolution experiments performed in section 6.2.2. Predictive accuracy, concept complexity and parameter complexity results are shown. The difference between these figures (apart from the particular results) and those given in section 6.2.2 as well as figures B.25 through B.28 is that fewer rules were assembled by the CORA algorithm. For each figure, the relevant problem and the number of assembled rules is given in parentheses after each figure caption.

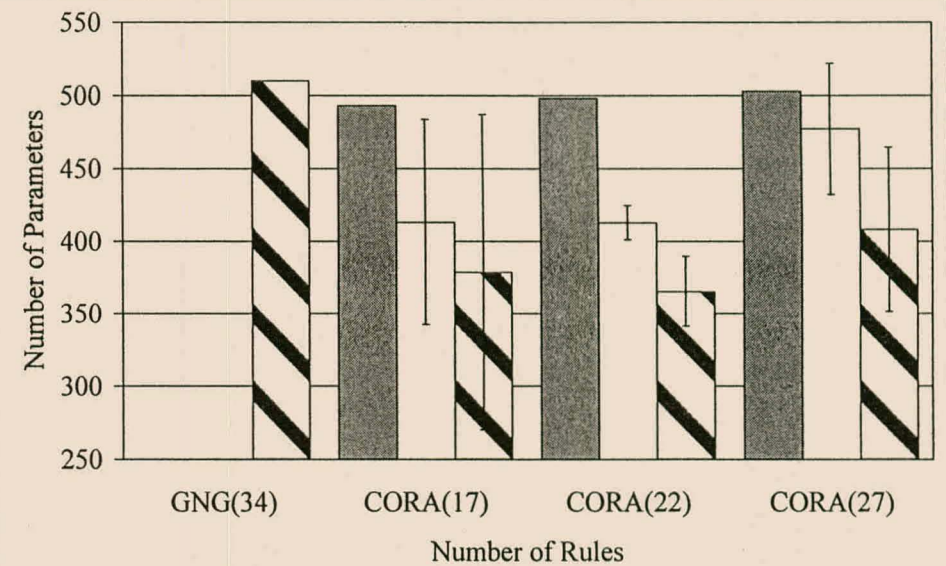
- Figures B.47 through B.53 give additional parameter complexity results for section 6.2.3. The efficacy of using swaps, moves or both swaps and moves to construct rule models is studied in section 6.2.3. The number in parentheses at the end of each figure caption is the number of rules used to construct the relevant rule models. In each figure caption, the abbreviation “SMB” stands for “swaps, moves or both swaps and moves”. The CORA algorithm assembled the maximum number of fuzzy rules (see section 5.3.3.2 for details) while generating these results.

Figures B.54 through B.89 present additional predictive accuracy and model complexity results for section 6.2.3. The CORA algorithm used fewer rules while generating these results, as indicated by the number in parentheses after each figure caption. A `minimum_swap_threshold` of 0.9 was used in all the experiments upon which figures B.47 through B.89 are based.

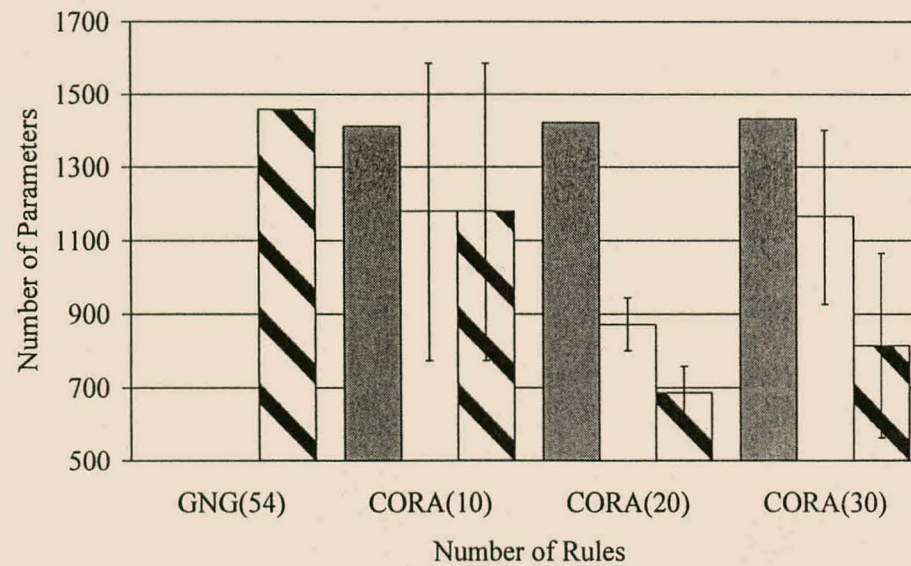
- Figures B.90 through B.95 give additional results for section 6.2.4 where the effect of consequent magnitude penalisation is studied. Specifically, parameter complexity results are presented for the problems that were investigated in section 6.2.4. In each figure the horizontal axis label for a particular subset of experimental runs gives the number of rules assembled in parentheses and whether consequent magnitude penalisation was employed or not (Yes or No). Note that consequent magnitude penalisation is referred to as weight penalisation in the figures.
- Figures B.96 through B.98 give additional results of the attribute space overlap experiments performed for section 6.2.5. In each figure the horizontal axis label for a particular subset of experimental runs gives the number of rules assembled in parentheses and whether attribute space overlap penalisation was used or not.
- Finally, figures B.99 and B.100 give parameter complexity results for the “minimisation of attribute space overlap” experiments performed for section 6.3.1.



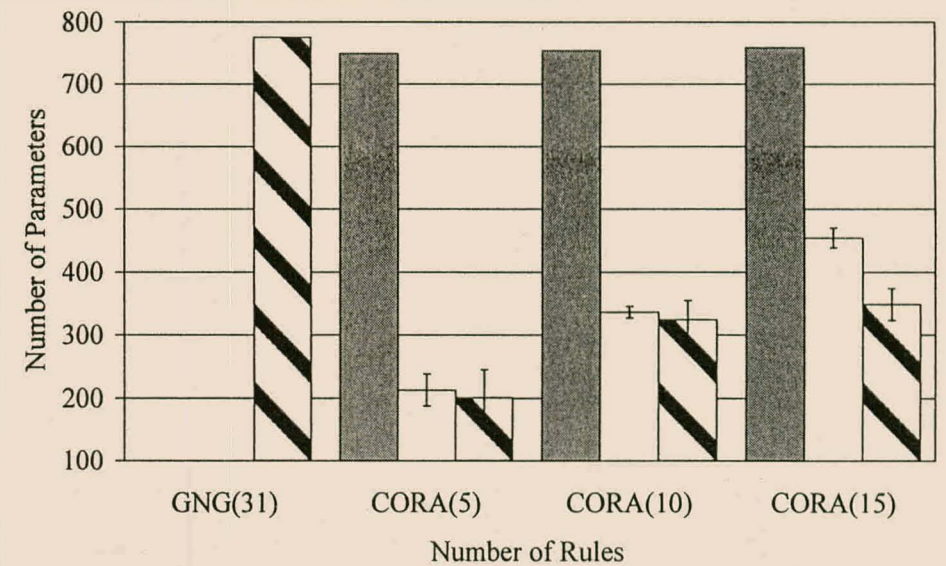
**Figure B.1 No. of Parameters vs. No. of Rules for Abalone Data**



**Figure B.2 No. of Parameters vs. No. of Rules for Auto Data**

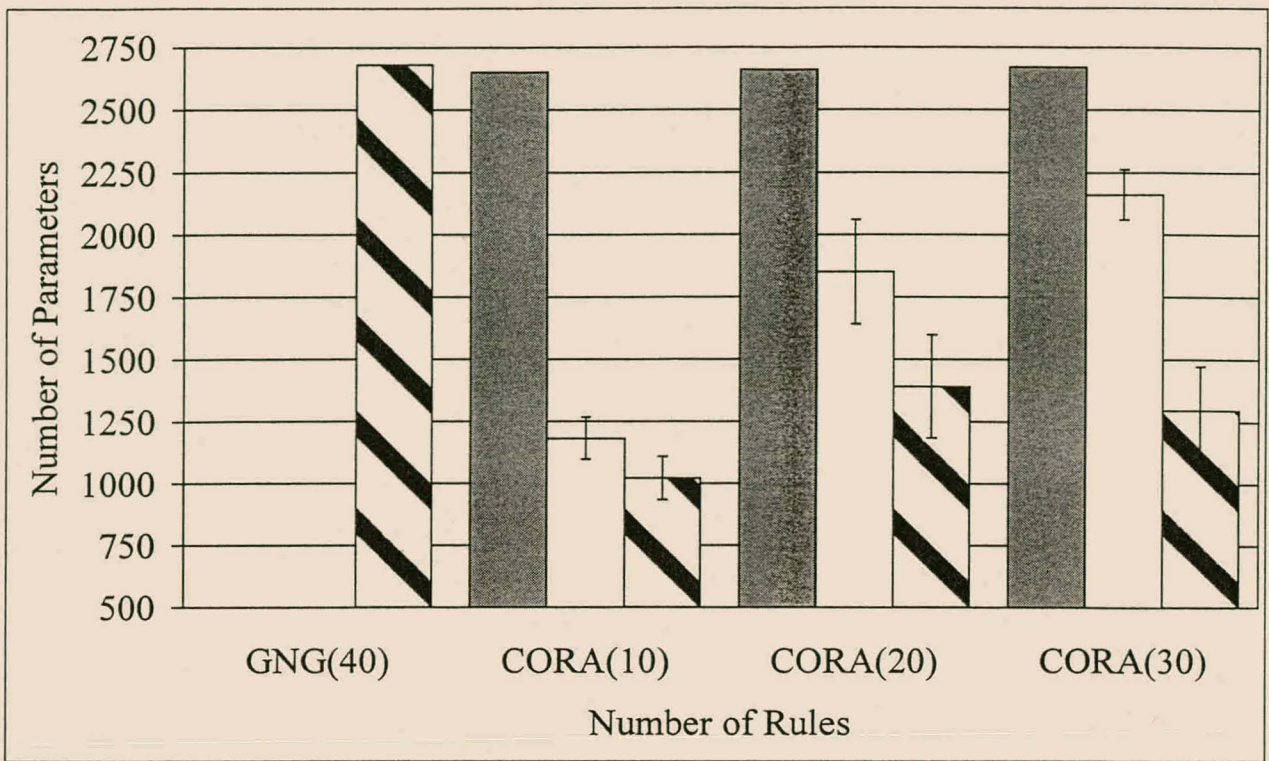


**Figure B.3 No. of Parameters vs. No. of Rules for Housing Data**

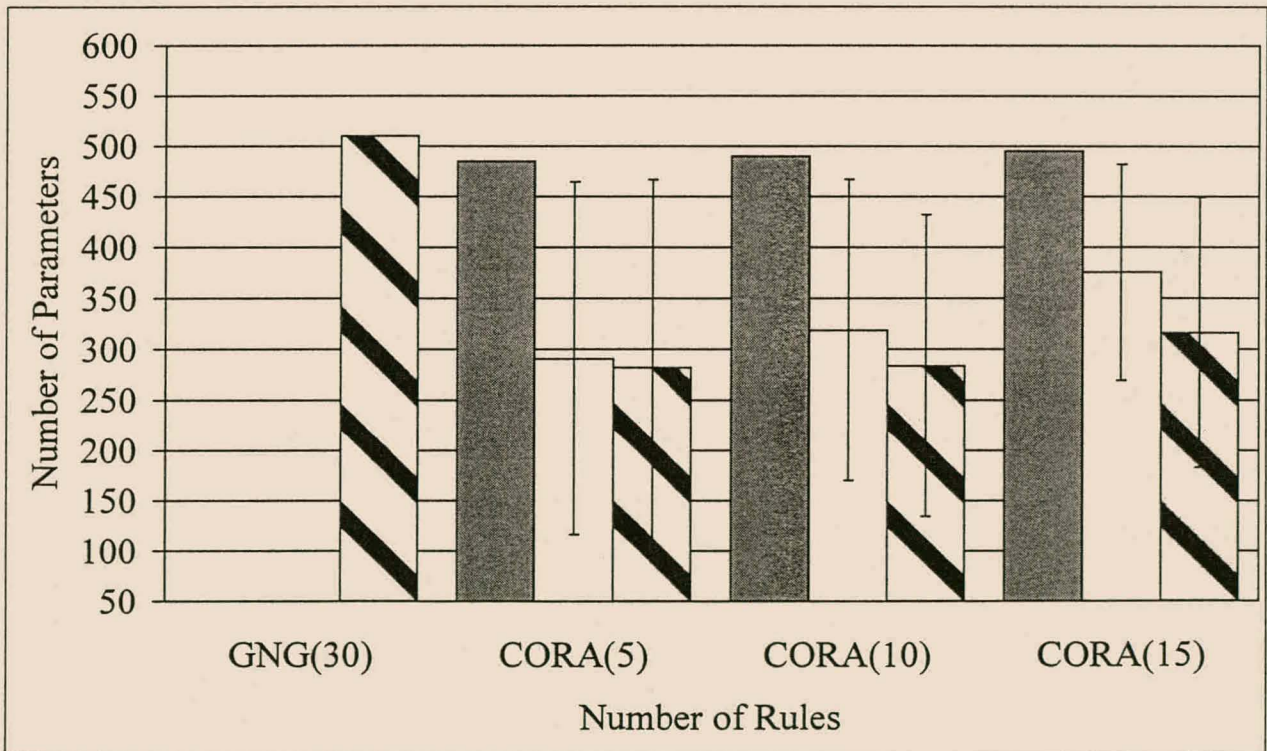


**Figure B.4 No. of Parameters vs. No. of Rules for Servo Problem**

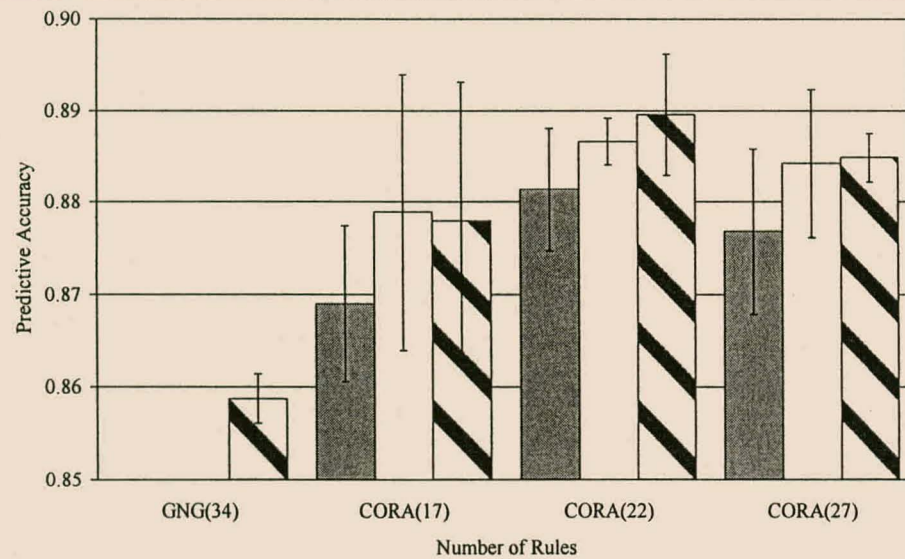




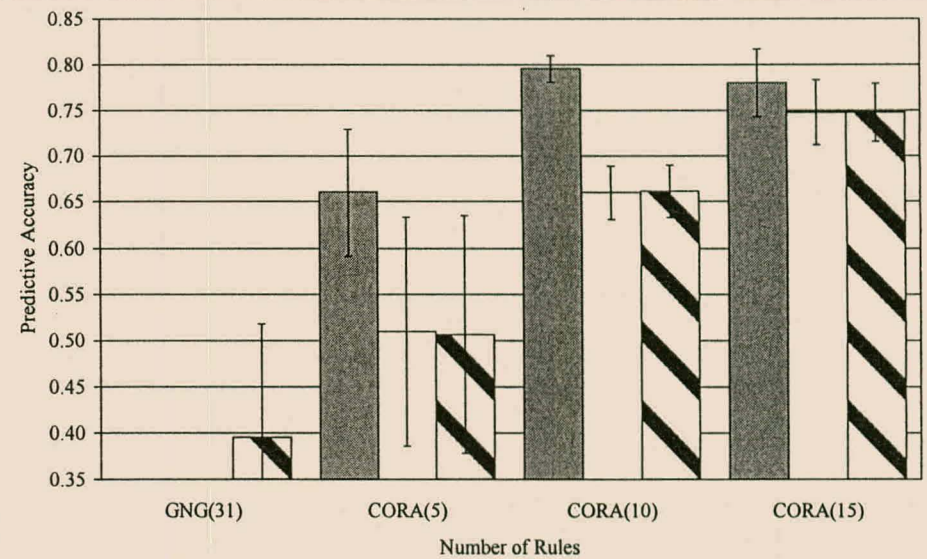
**Figure B.5** No. of Parameters vs. No. of Rules for Ionosphere Data



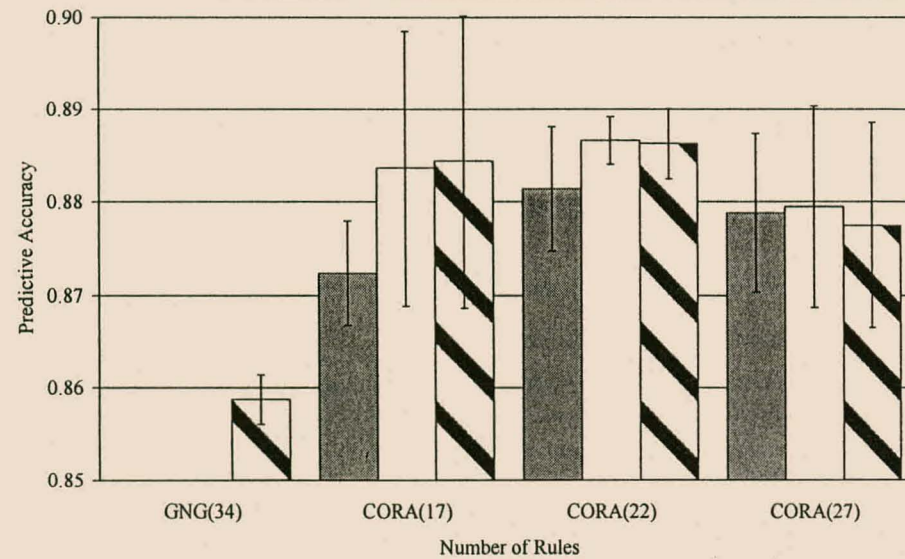
**Figure B.6** No. of Parameters vs. No. of Rules for Pima Problem



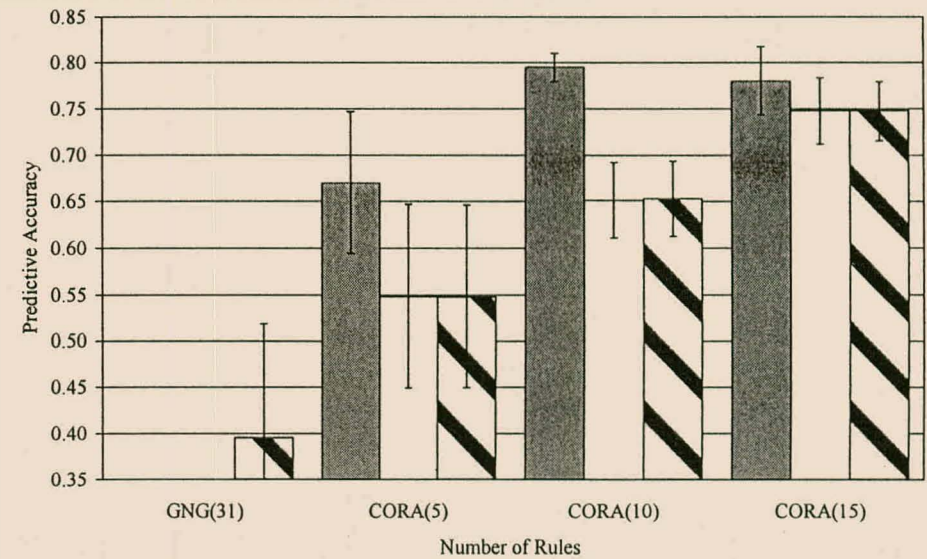
**Figure B.7 Accuracy vs. No. of Rules for Auto Problem (0.5)**



**Figure B.8 Accuracy vs. No. of Rules for Servo Problem (0.5)**

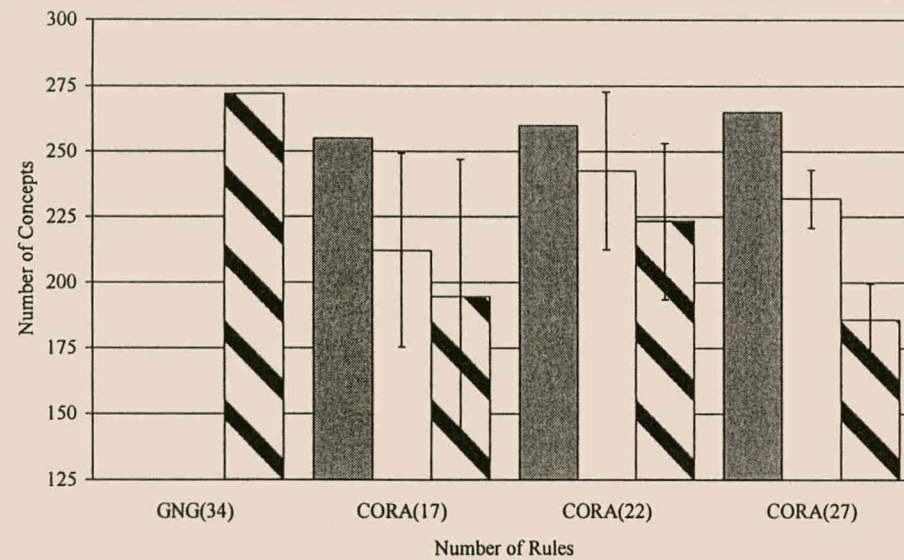


**Figure B.9 Accuracy vs. No. of Rules for Auto Problem (0.01)**

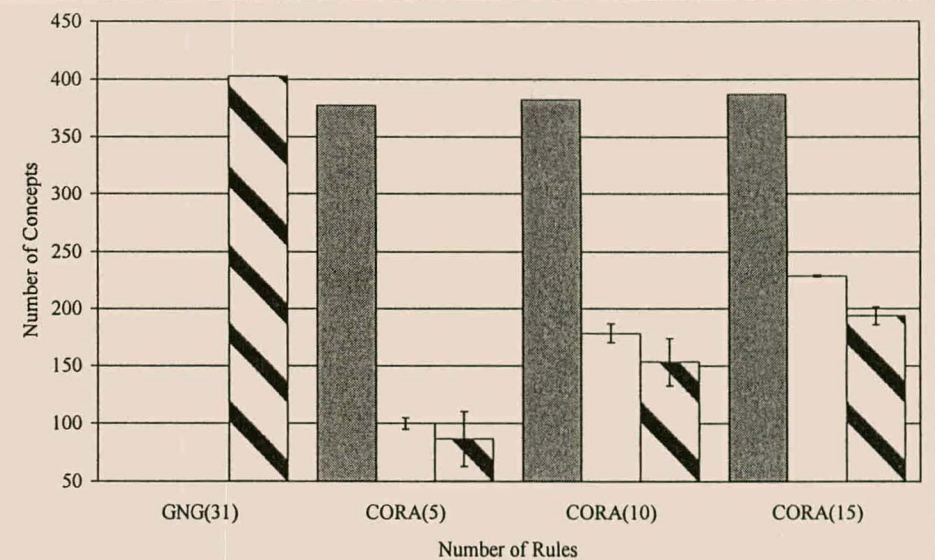


**Figure B.10 Accuracy vs. No. of Rules for Servo Problem (0.01)**

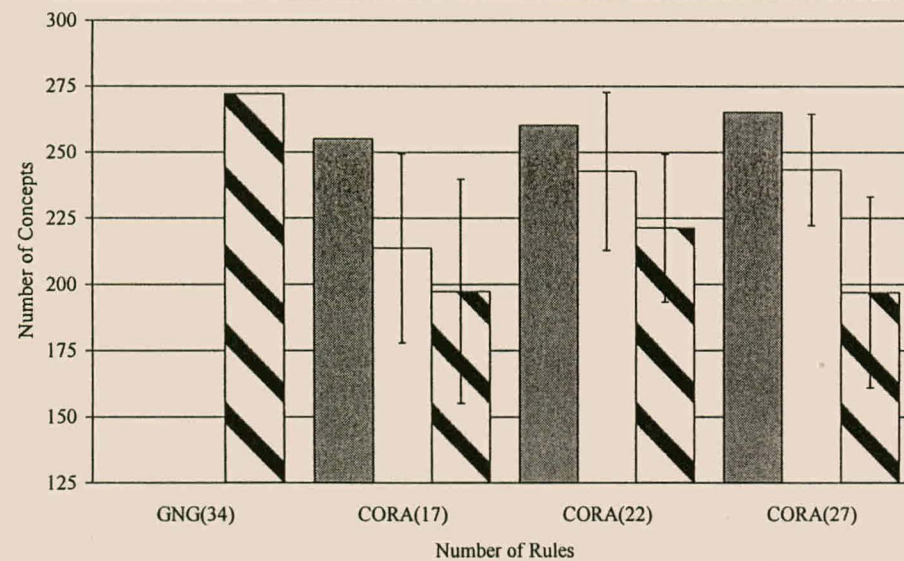




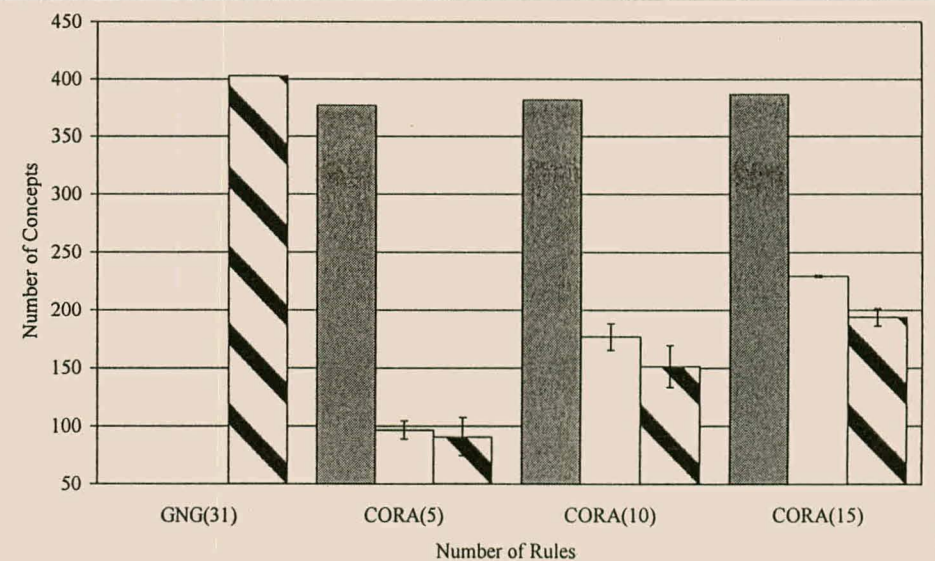
**Figure B.11 No. of Concepts vs. No. of Rules for Auto Data (0.5)**



**Figure B.12 No. of Concepts vs. No. of Rules for Servo Data (0.5)**

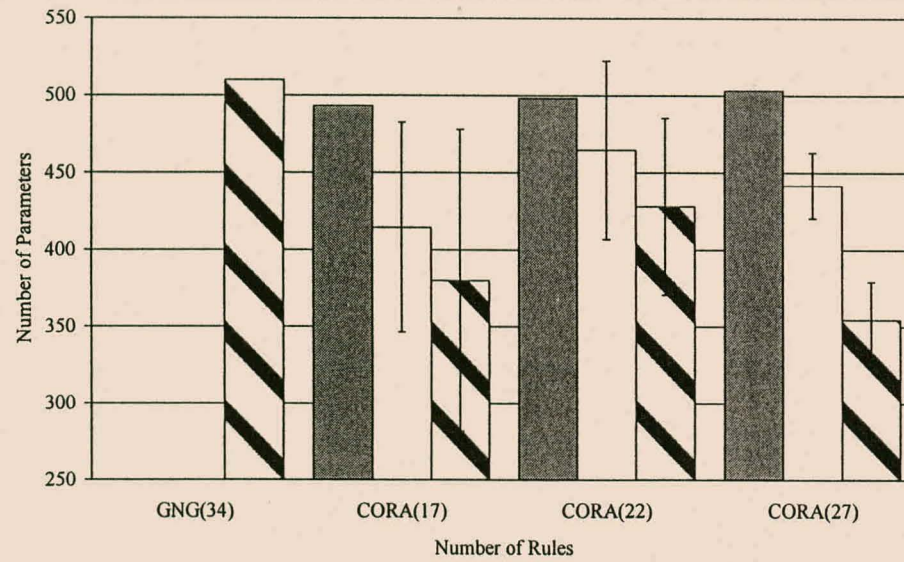


**Figure B.13 No. of Concepts vs. No. of Rules for Auto Data (0.01)**

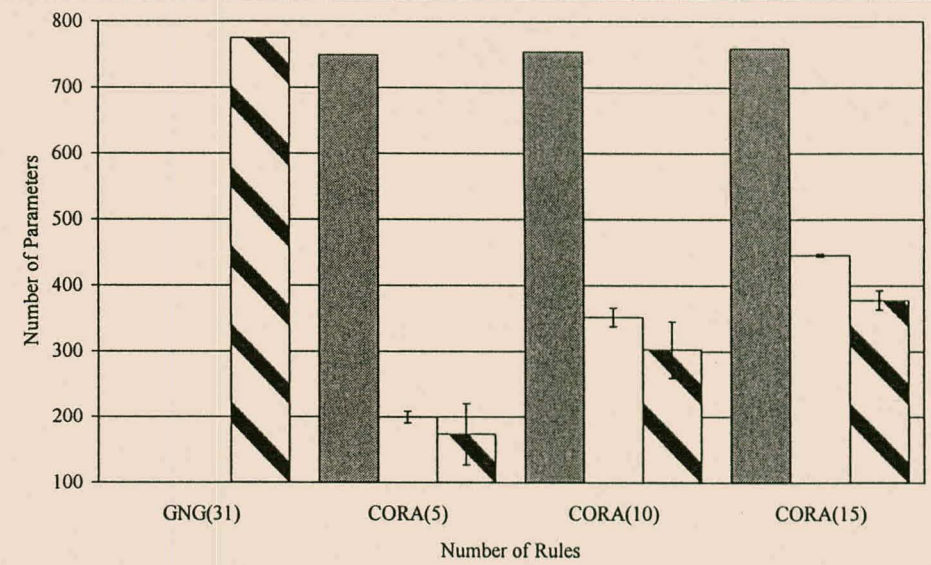


**Figure B.14 No. of Concepts vs. No. of Rules for Servo Data (0.01)**

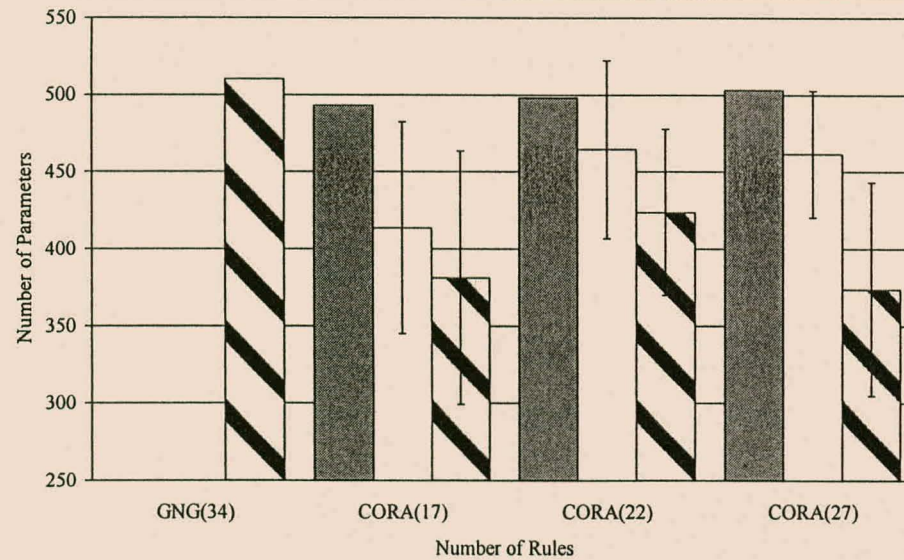




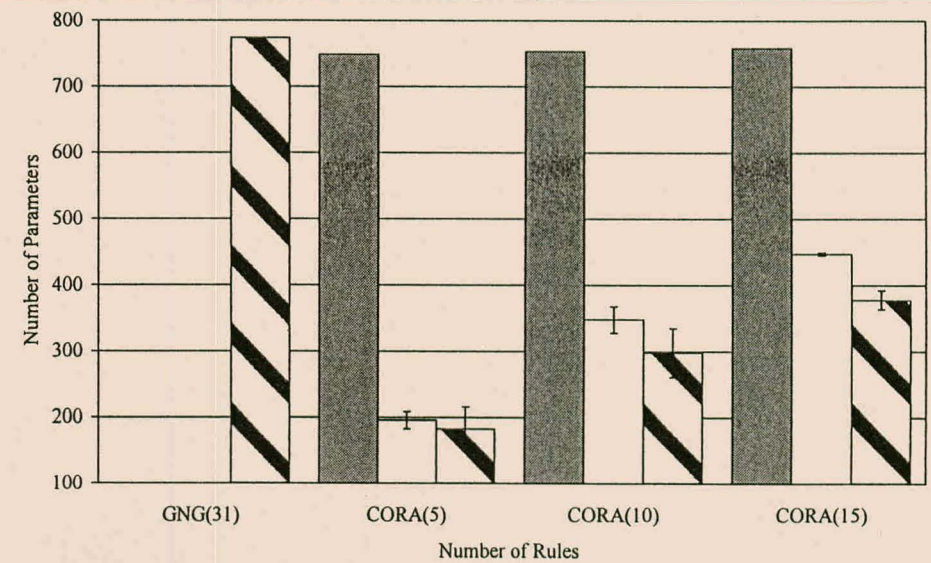
**Figure B.15 No. of Parameters vs. No. of Rules for Auto Data (0.5)**



**Figure B.16 No. of Parameters vs. No. of Rules for Servo Data (0.5)**

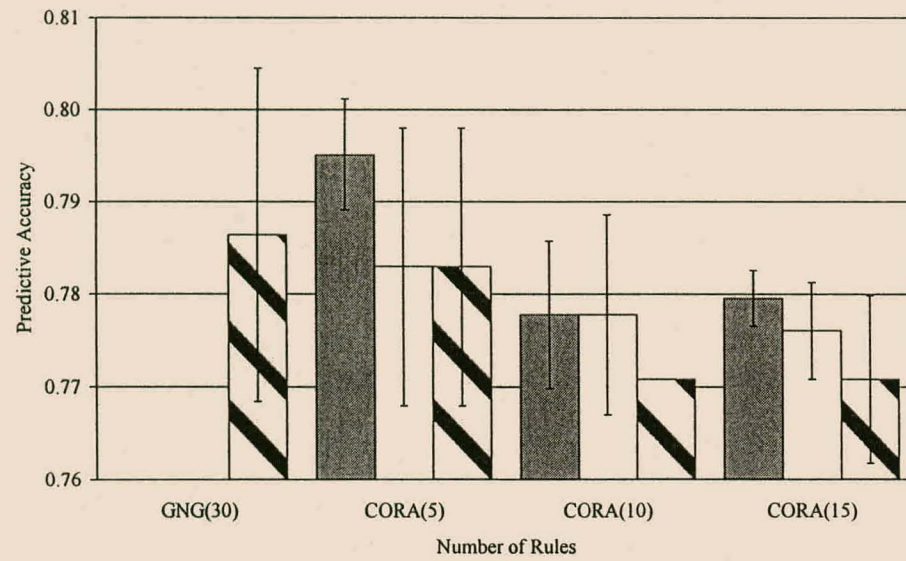


**Figure B.17 No. of Parameters vs. No. of Rules for Auto Data (0.01)**

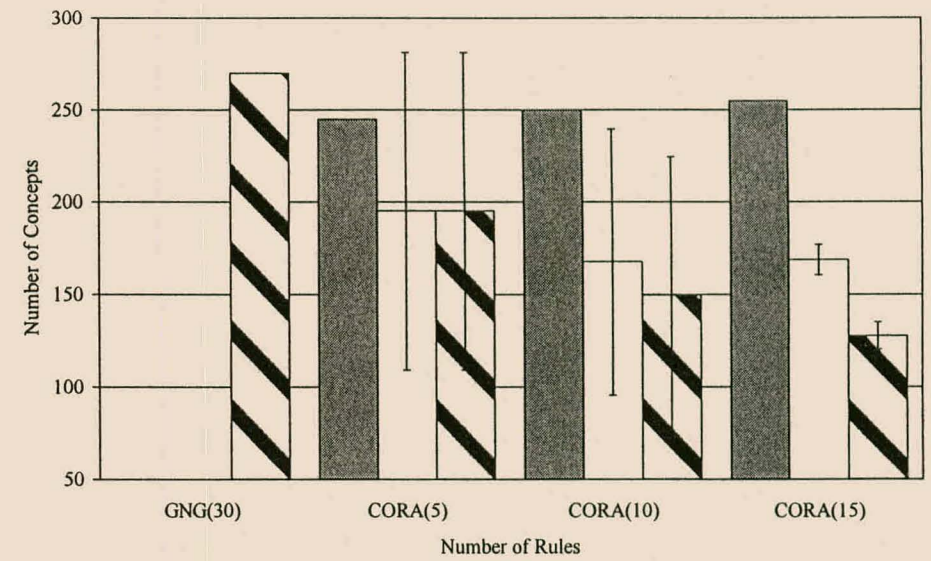


**Figure B.18 Parameters vs. No. of Rules for Servo Data (0.01)**

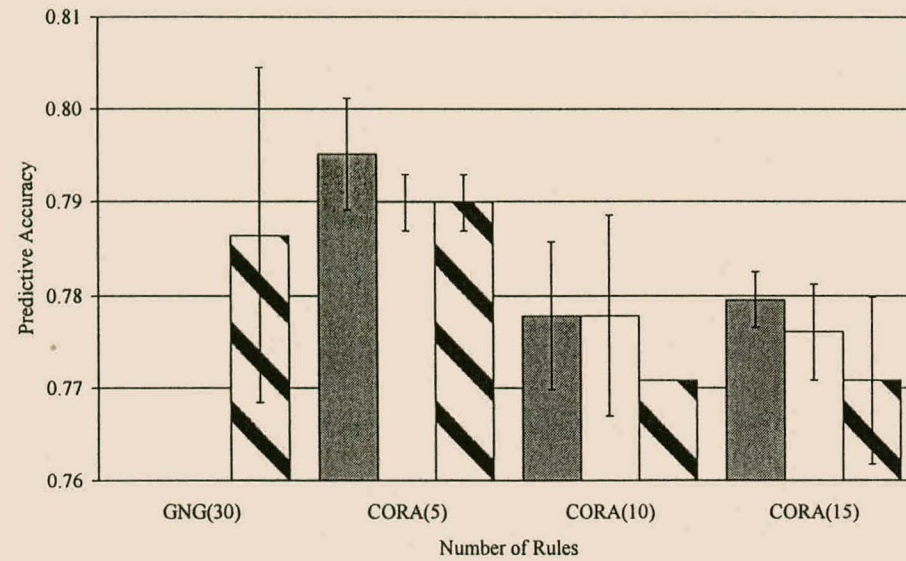




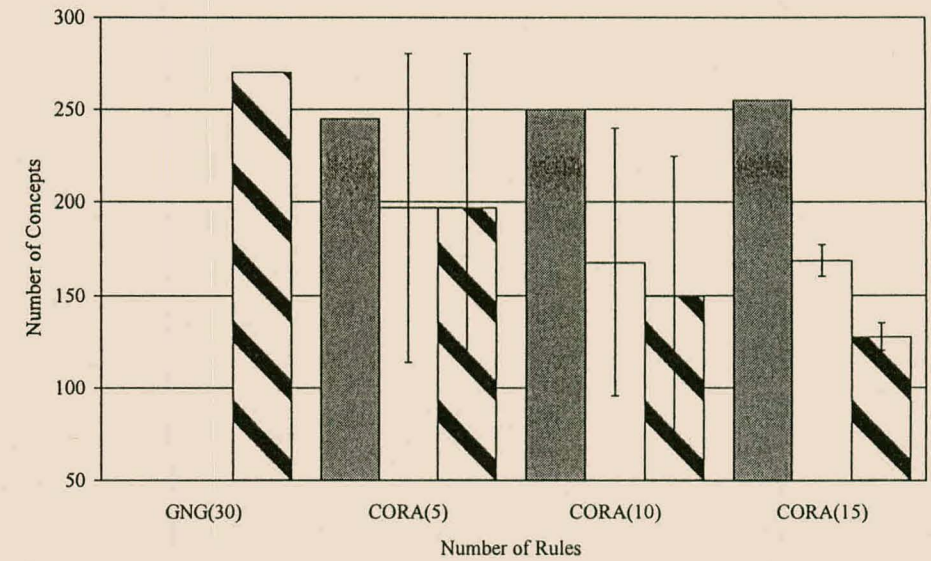
**Figure B.19 Accuracy vs. No. of Rules for Pima Problem (0.5)**



**Figure B.20 No. of Concepts vs. No. of Rules for Pima Data (0.5)**



**Figure B.21 Accuracy vs. No. of Rules for Pima Problem (0.01)**



**Figure B.22 No. of Concepts vs. No. of Rules for Pima Data (0.01)**

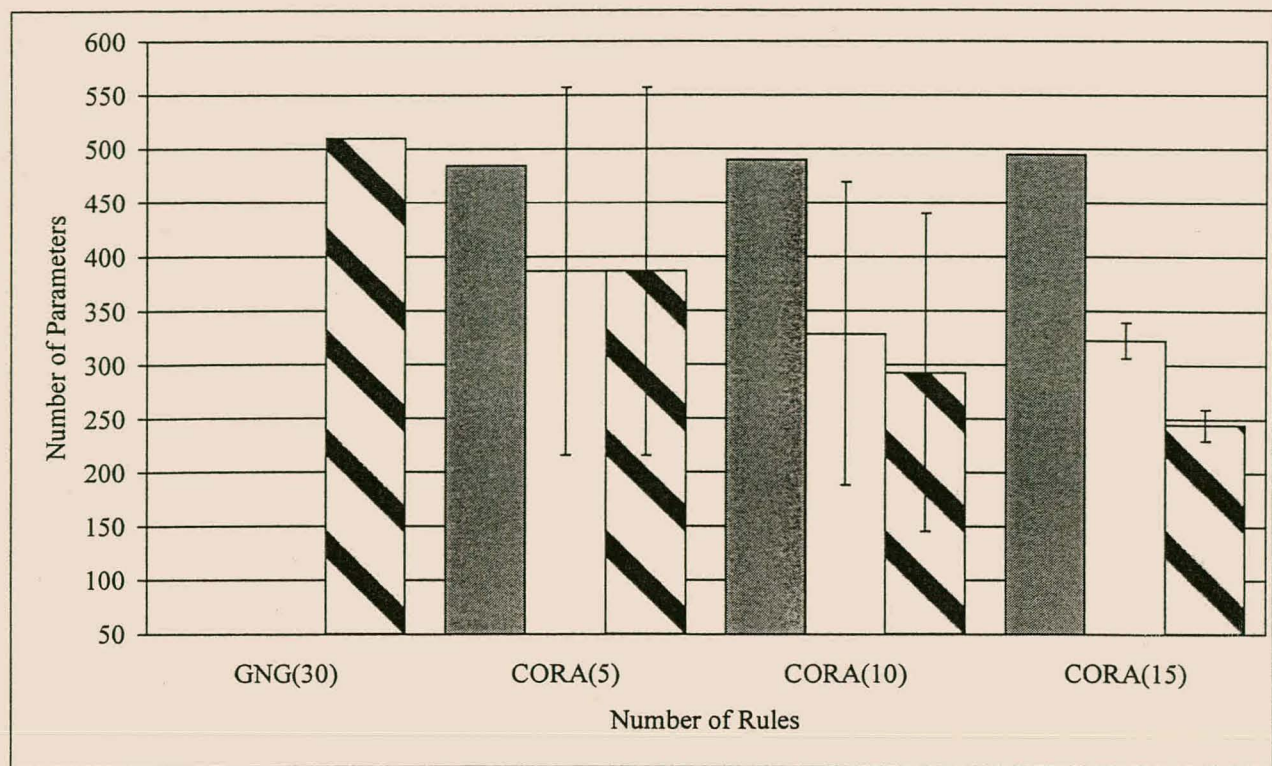


Figure B.23 No. of Parameters vs. No. of Rules for Pima Problem (0.5)

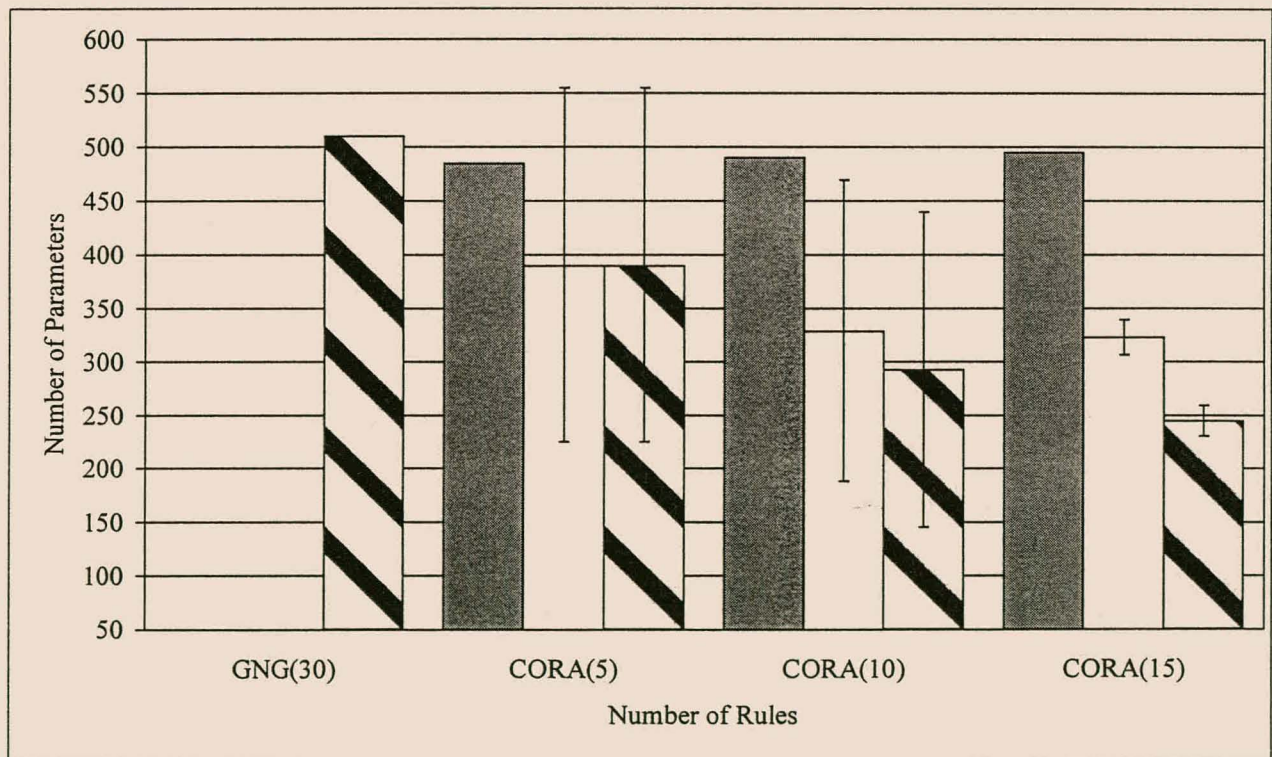
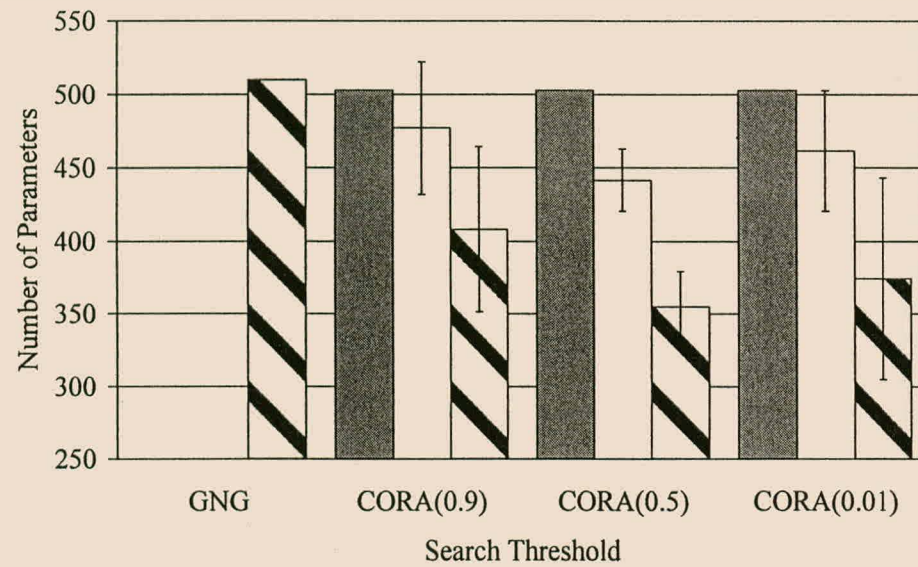
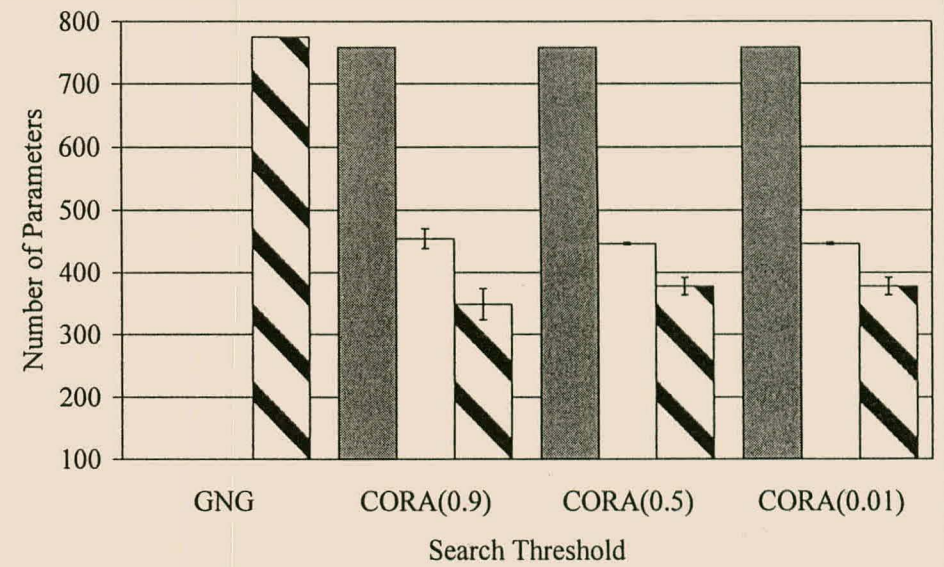


Figure B.24 No. of Parameters vs. No. of Rules for Pima Problem (0.01)

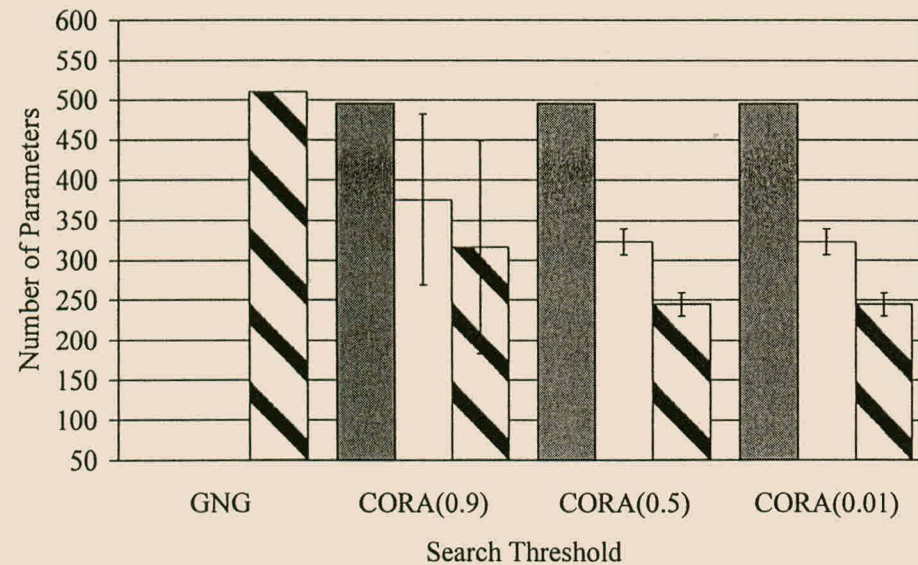




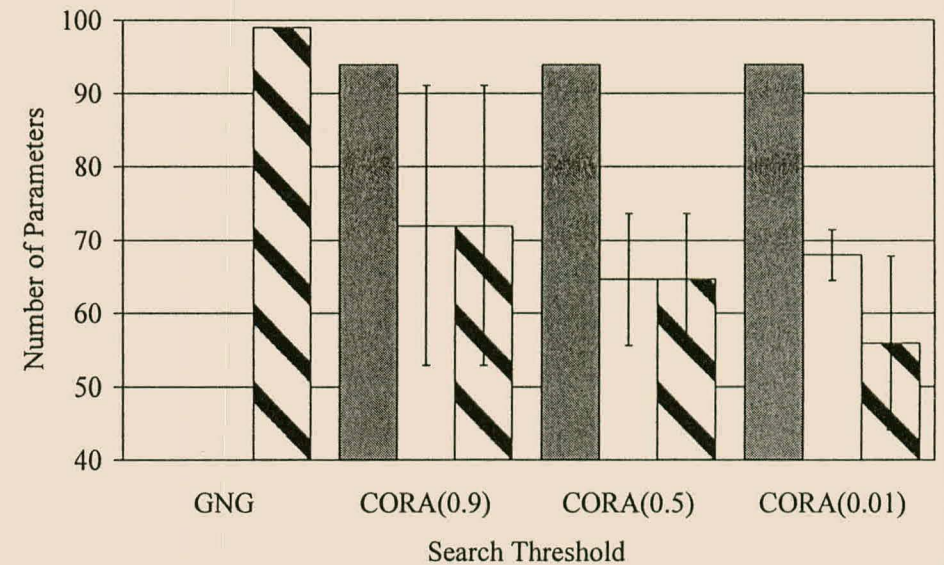
**Figure B.25 No. of Parameters vs. Search Resolution (Auto (27))**



**Figure B.26 No. of Parameters vs. Search Resolution (Servo (15))**

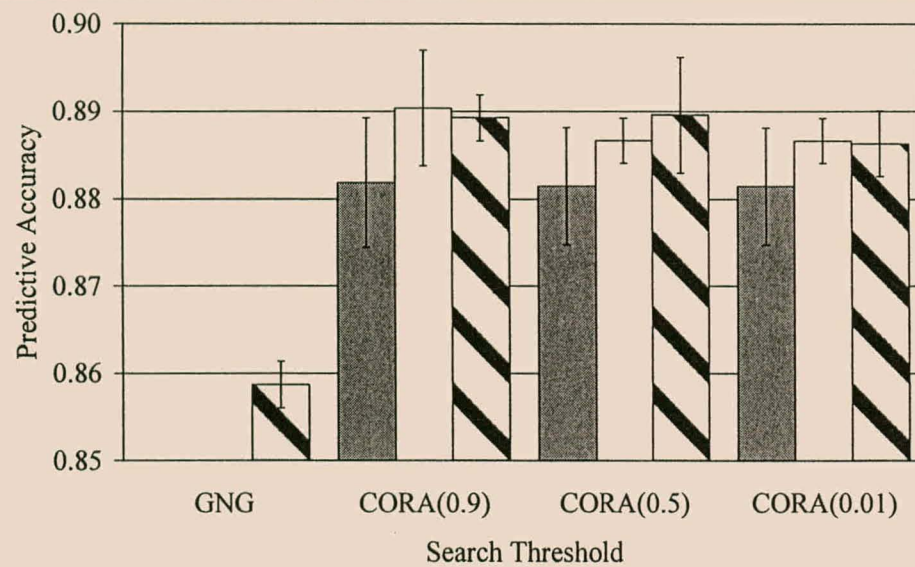


**Figure B.27 No. of Parameters vs. Search Resolution (Pima (15))**

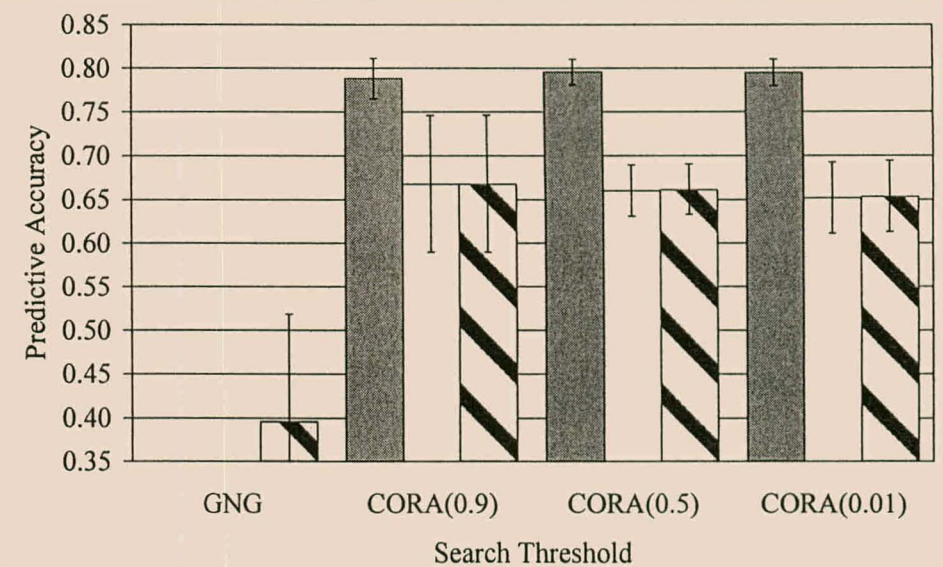


**Figure B.28 Parameters vs. Search Resolution (Slugflow (6))**

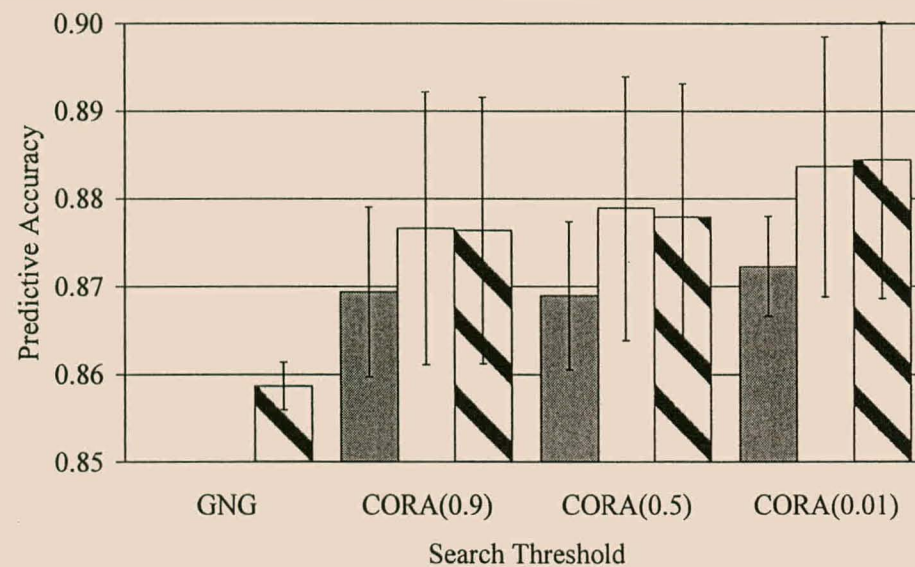




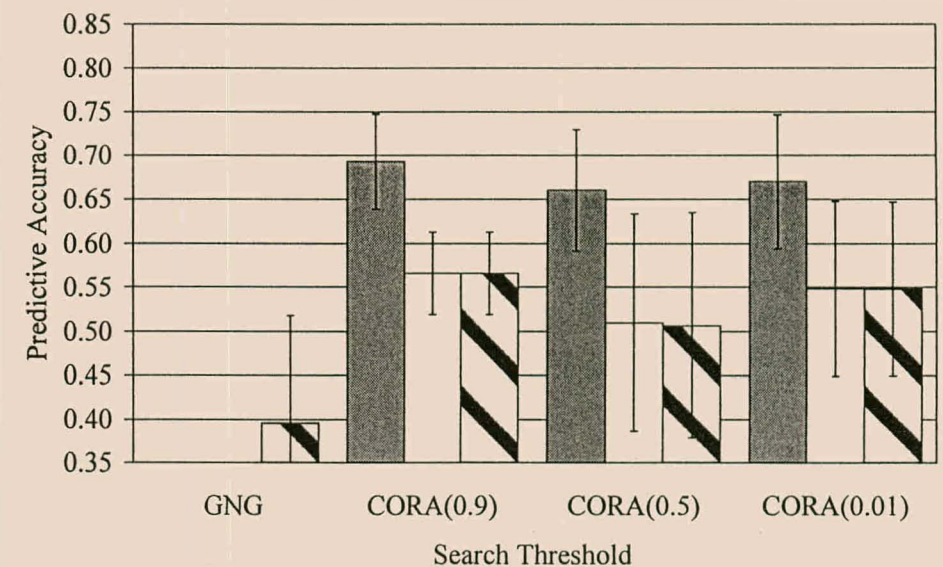
**Figure B.29 Accuracy vs. Search Resolution (Auto (22))**



**Figure B.30 Accuracy vs. Search Resolution (Servo (10))**

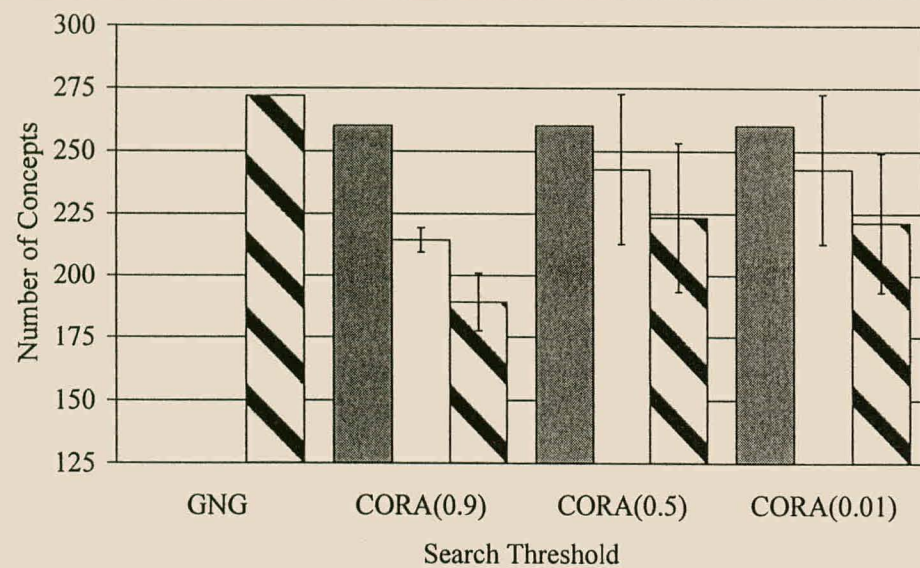


**Figure B.31 Accuracy vs. Search Resolution (Auto (17))**

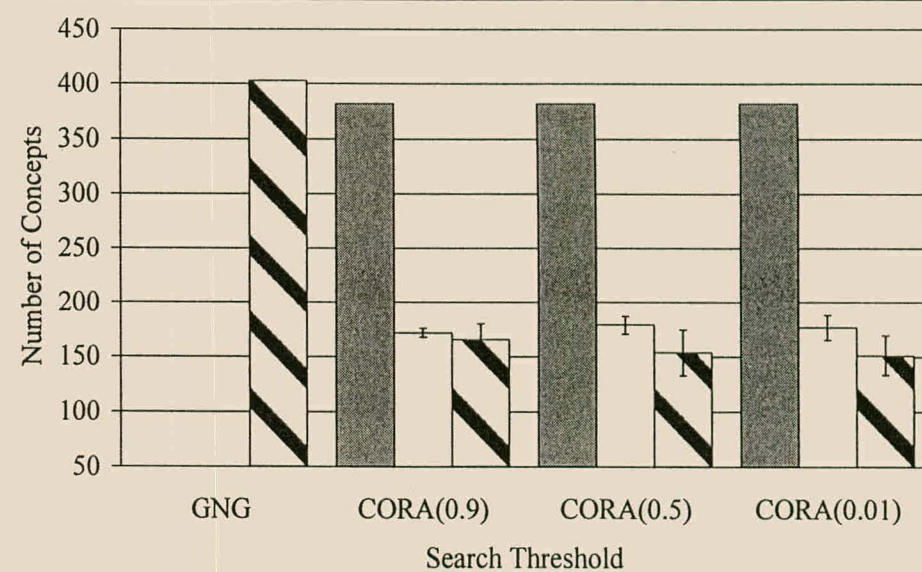


**Figure B.32 Accuracy vs. Search Resolution (Servo (5))**

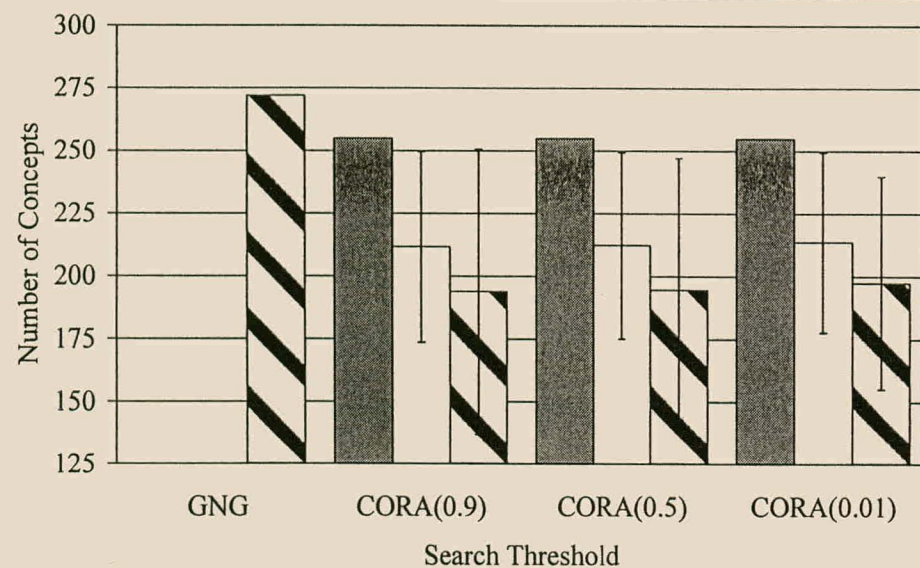




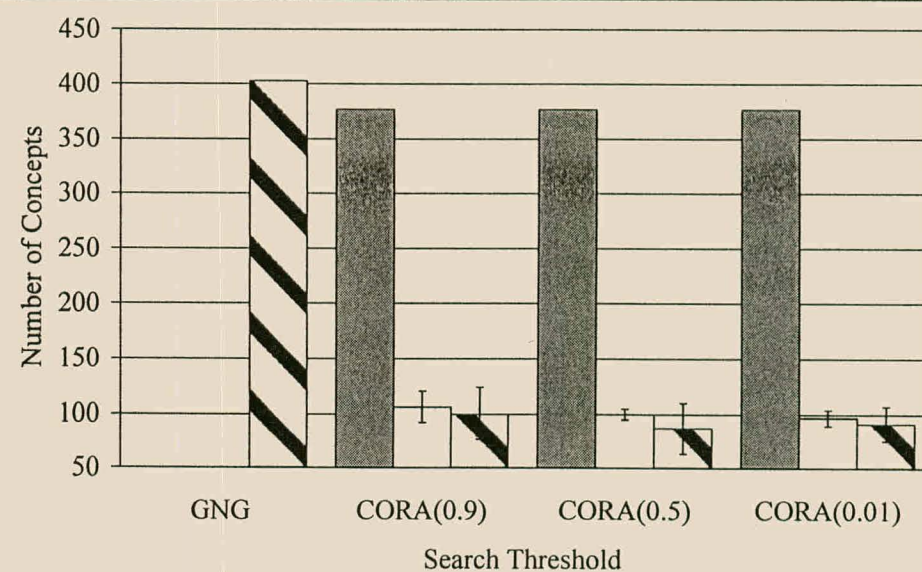
**Figure B.33 No. of Concepts vs. Search Resolution (Auto (22))**



**Figure B.34 No. of Concepts vs. Search Resolution (Servo (10))**

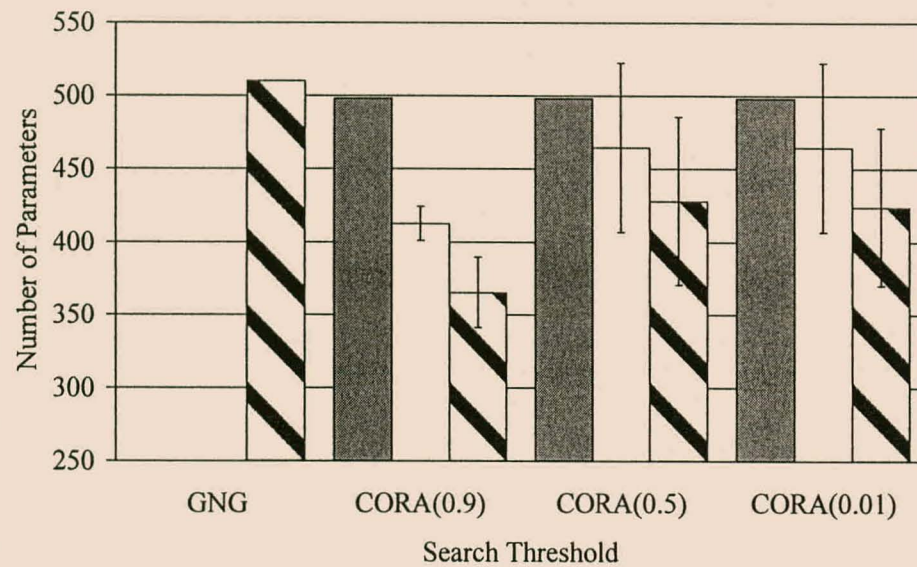


**Figure B.35 No. of Concepts vs. Search Resolution (Auto (17))**

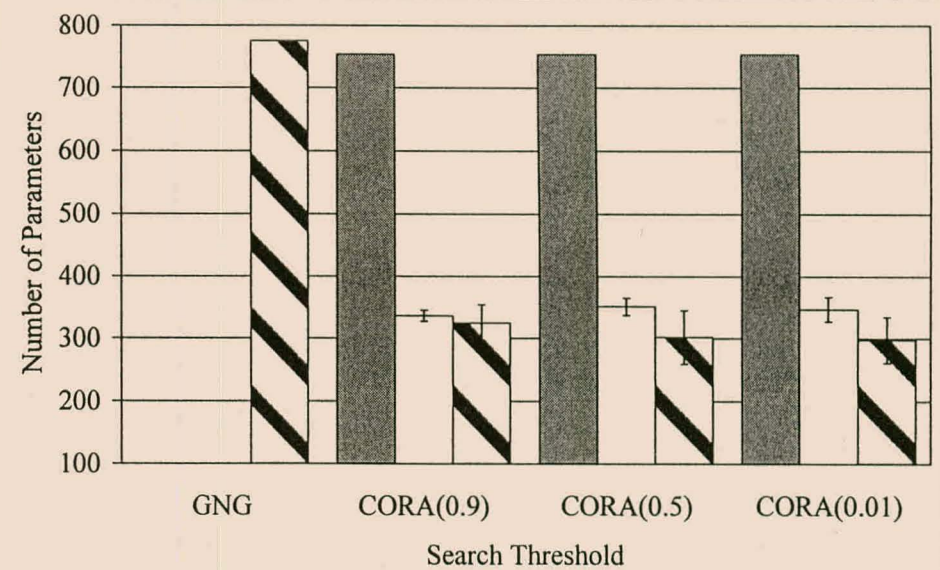


**Figure B.36 No. of Concepts vs. Search Resolution (Servo (5))**

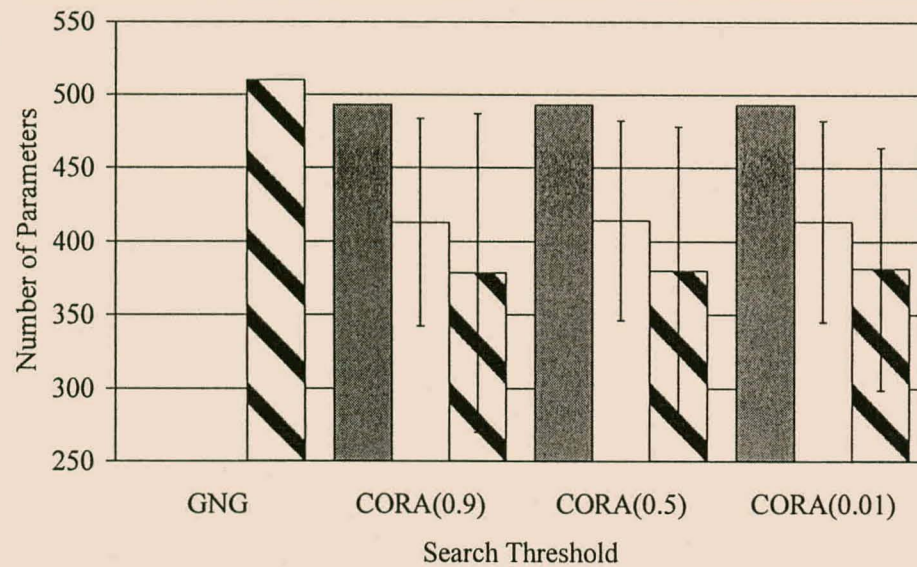




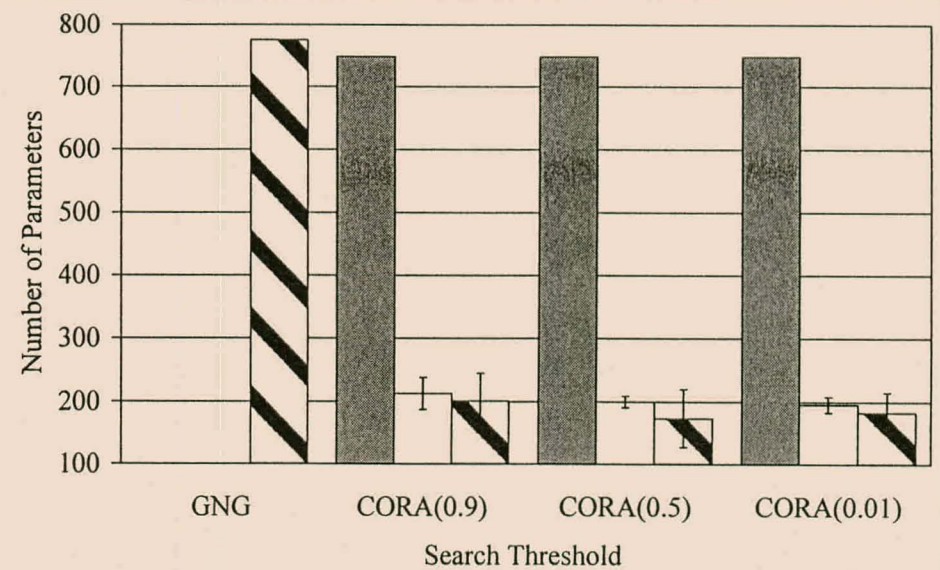
**Figure B.37 No. of Parameters vs. Search Resolution (Auto (22))**



**Figure B.38 No. of Parameters vs. Search Resolution (Servo (10))**



**Figure B.39 No. of Parameters vs. Search Resolution (Auto (17))**



**Figure B.40 No. of Parameters vs. Search Resolution (Servo (5))**



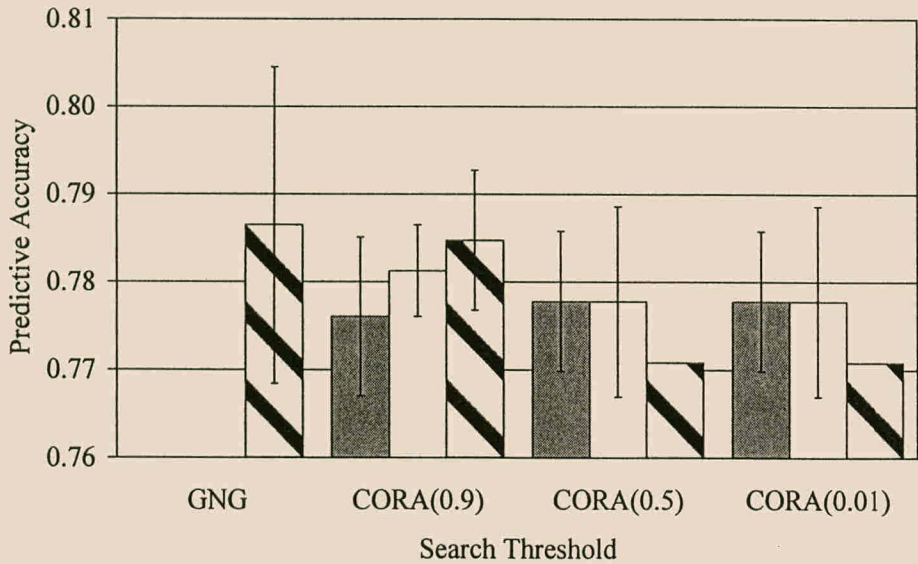


Figure B.41 Accuracy vs. Search Resolution (Pima (10))

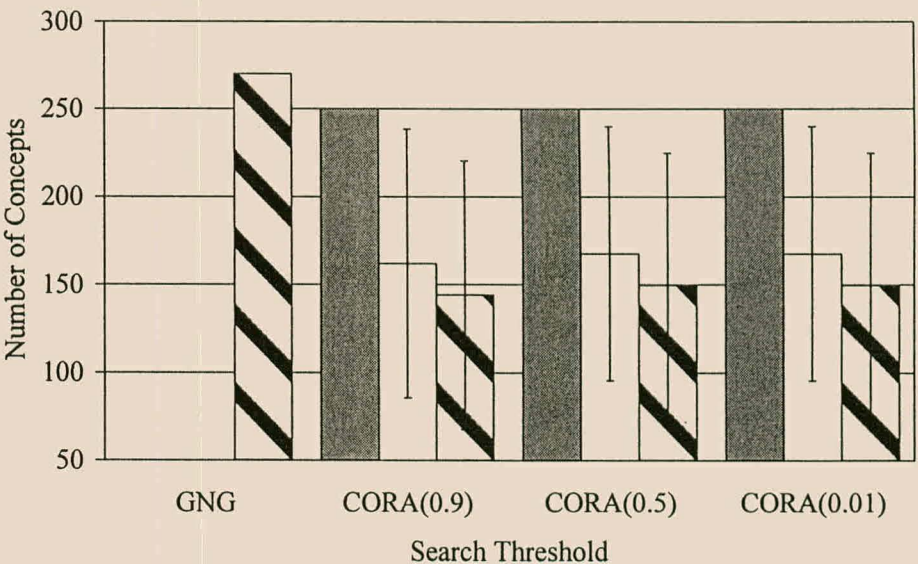


Figure B.42 No. of Concepts vs. Search Resolution (Pima (10))

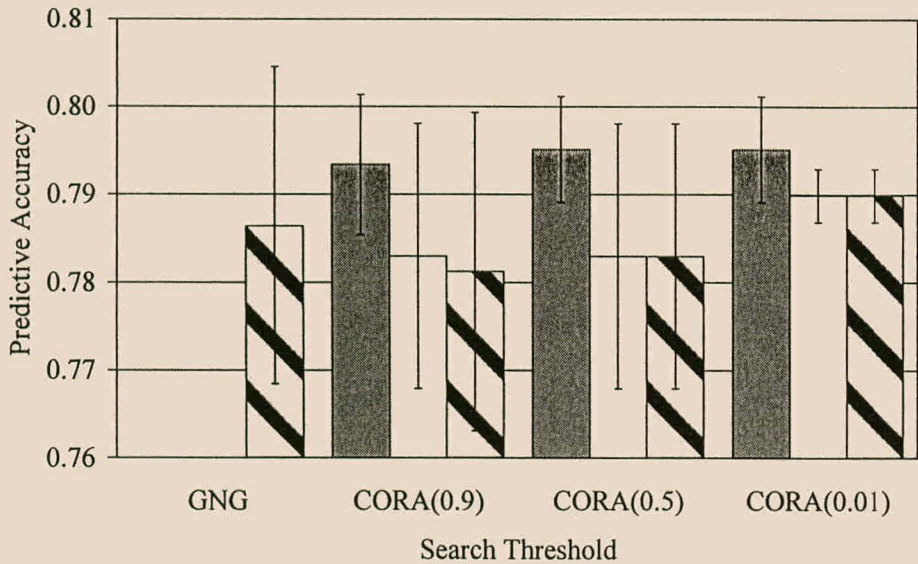


Figure B.43 Accuracy vs. Search Resolution (Pima (5))

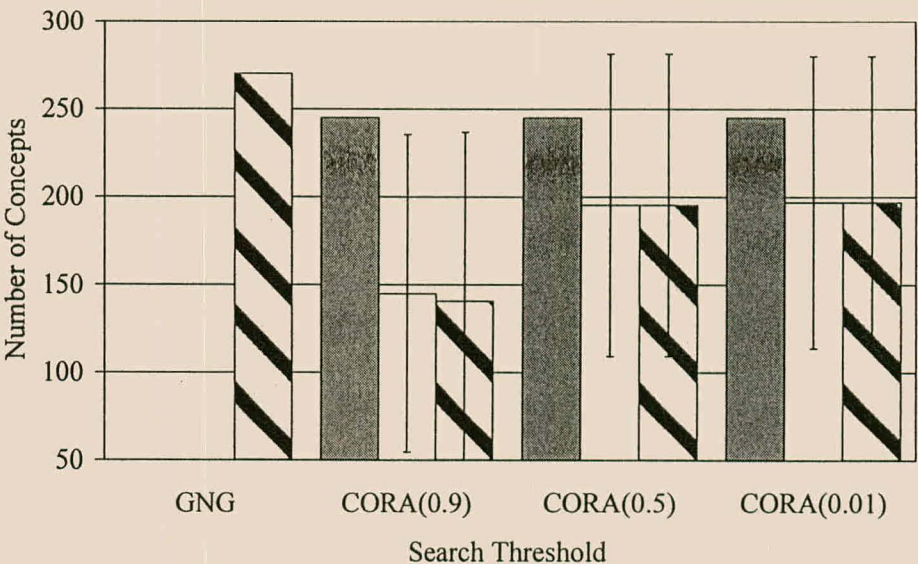


Figure B.44 No. of Concepts vs. Search Resolution (Pima (5))

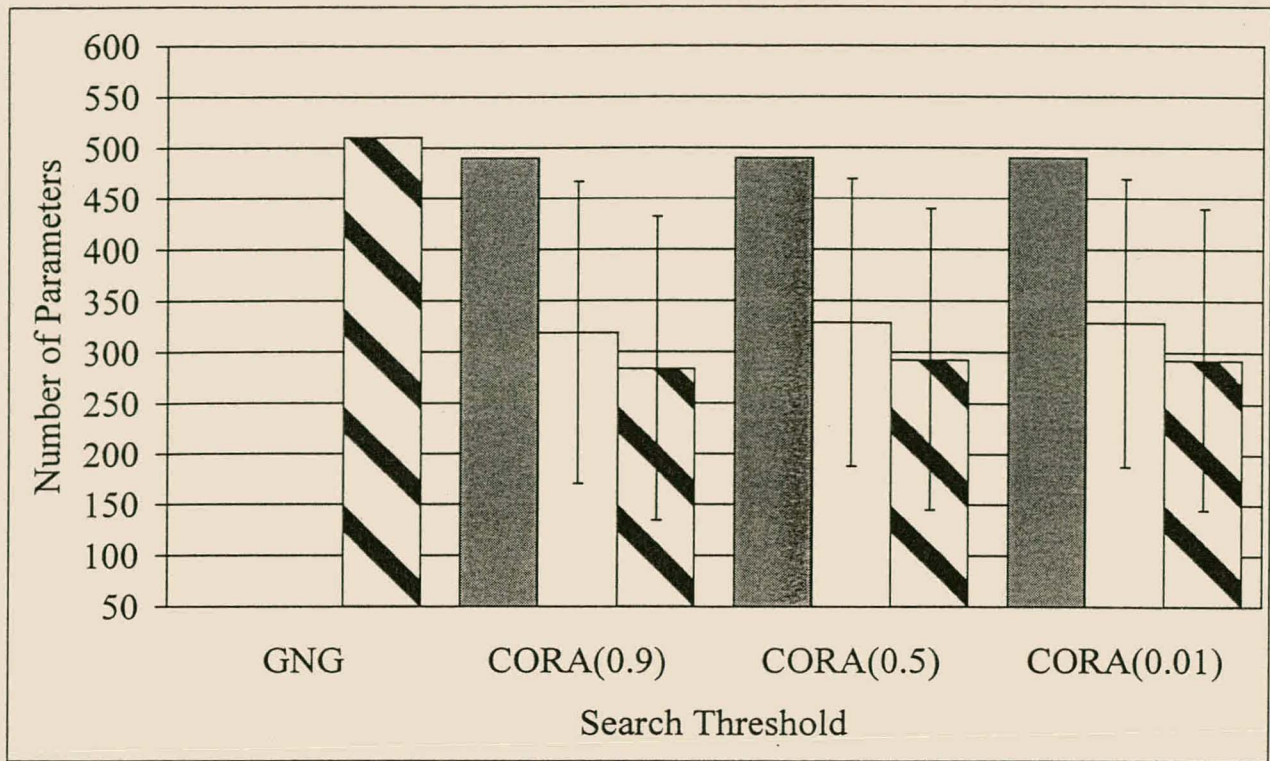


Figure B.45 No. of Parameters vs. Search Resolution (Pima (10))

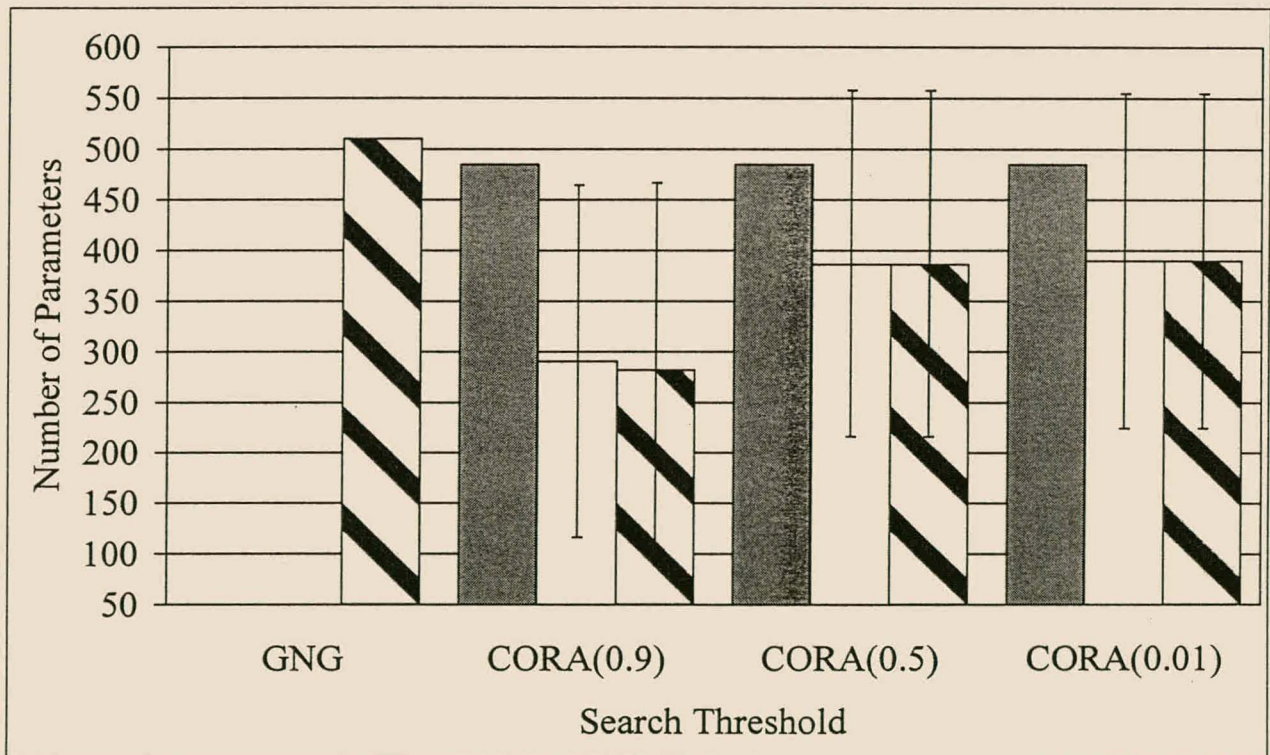
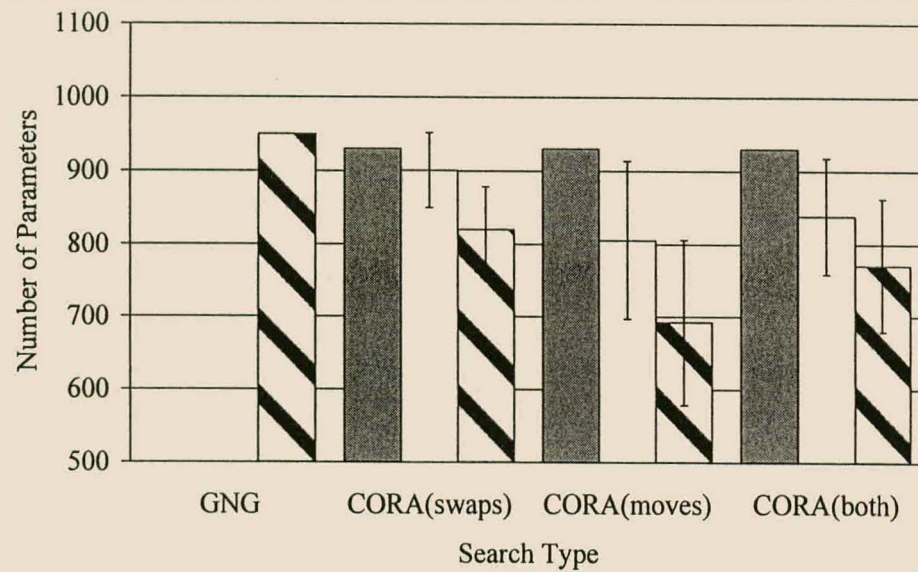
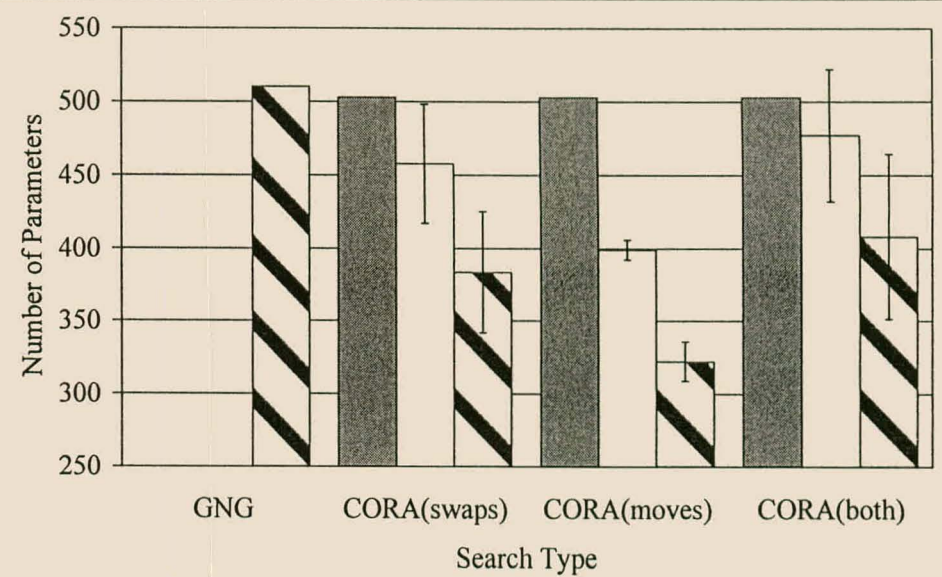


Figure B.46 No. of Parameters vs. Search Resolution (Pima (5))

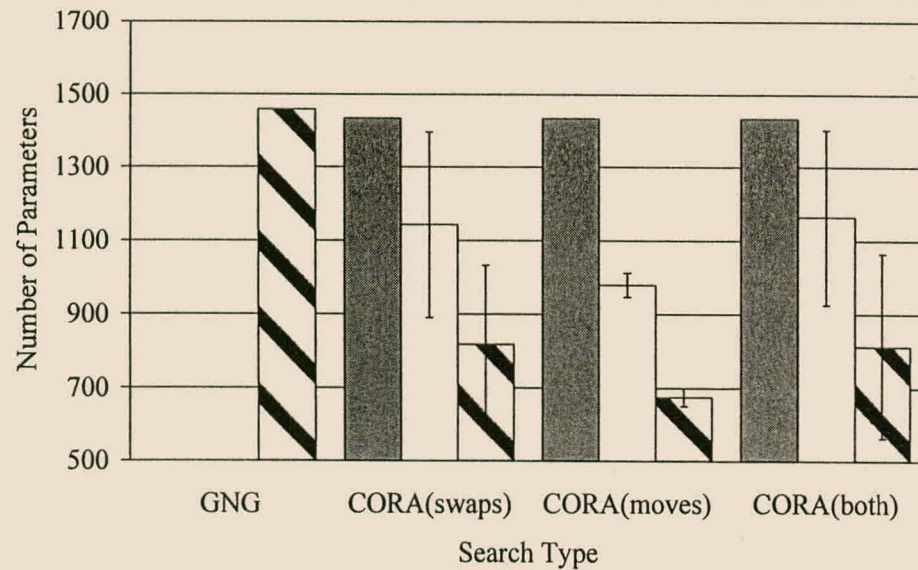




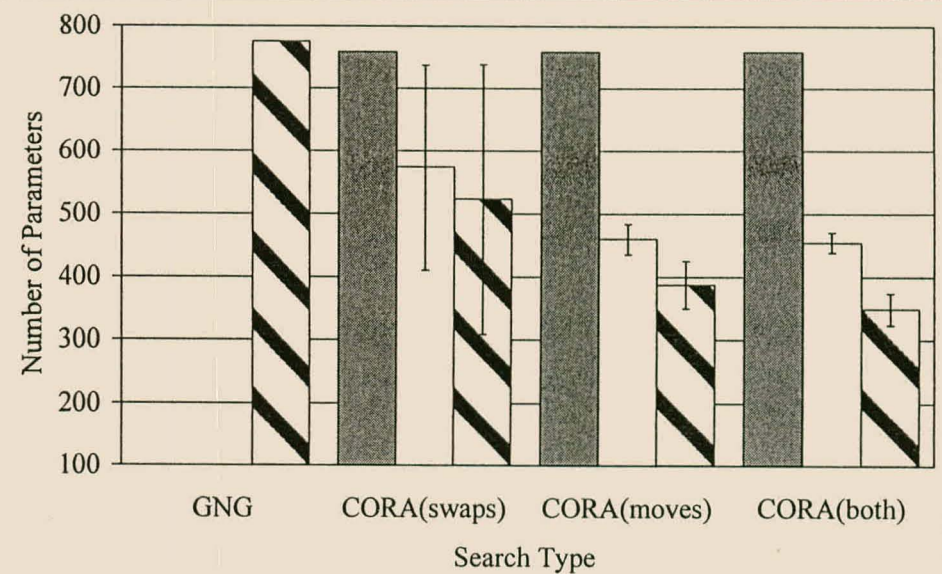
**Figure B.47 No. of Parameters vs. SMB for Abalone Data (30)**



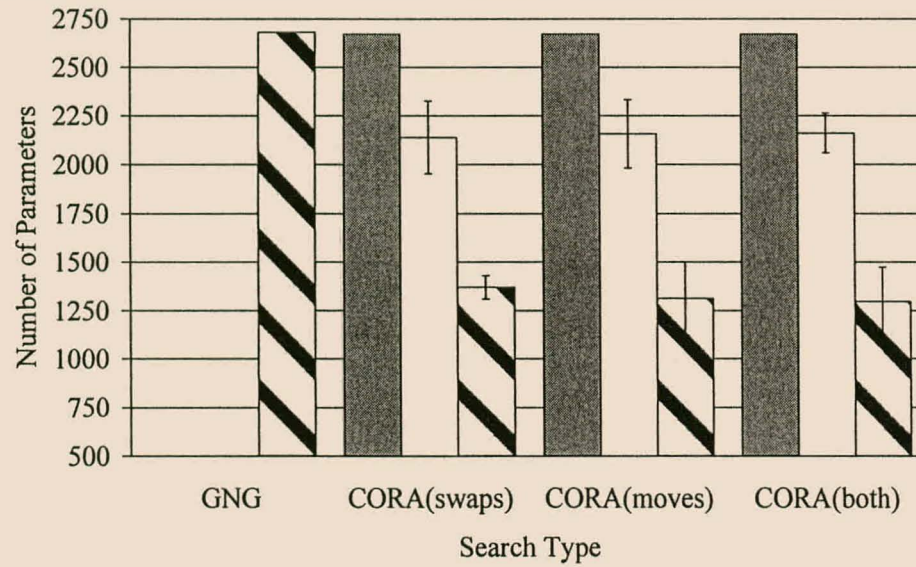
**Figure B.48 No. of Parameters vs. SMB for Auto Data (27)**



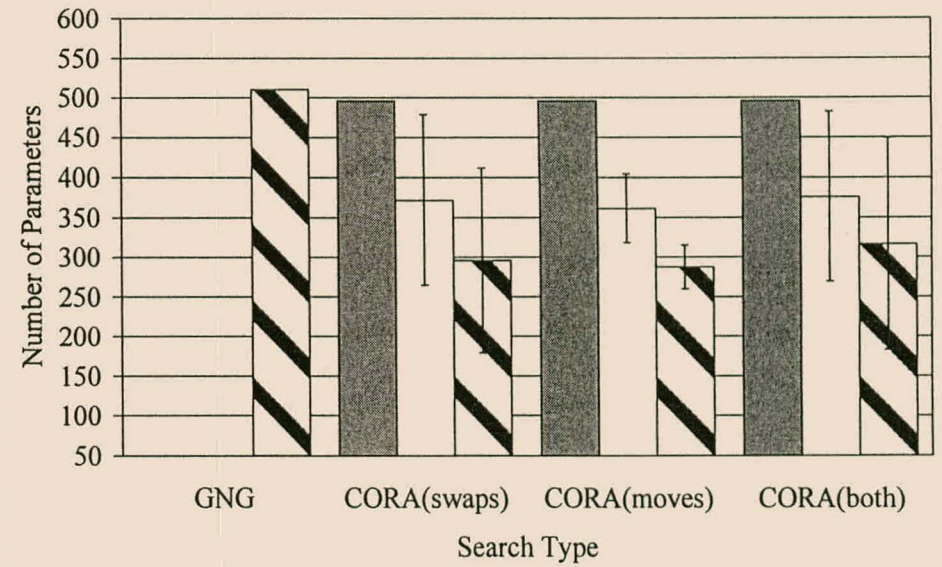
**Figure B.49 No. of Parameters vs. SMB for Housing Data (30)**



**Figure B.50 Parameters vs. SMB for Servo Problem (15)**



**Figure B.51 No. of Parameters vs. SMB for Ionosphere Data (30)**



**Figure B.52 No. of Parameters vs. SMB for Pima Data (15)**



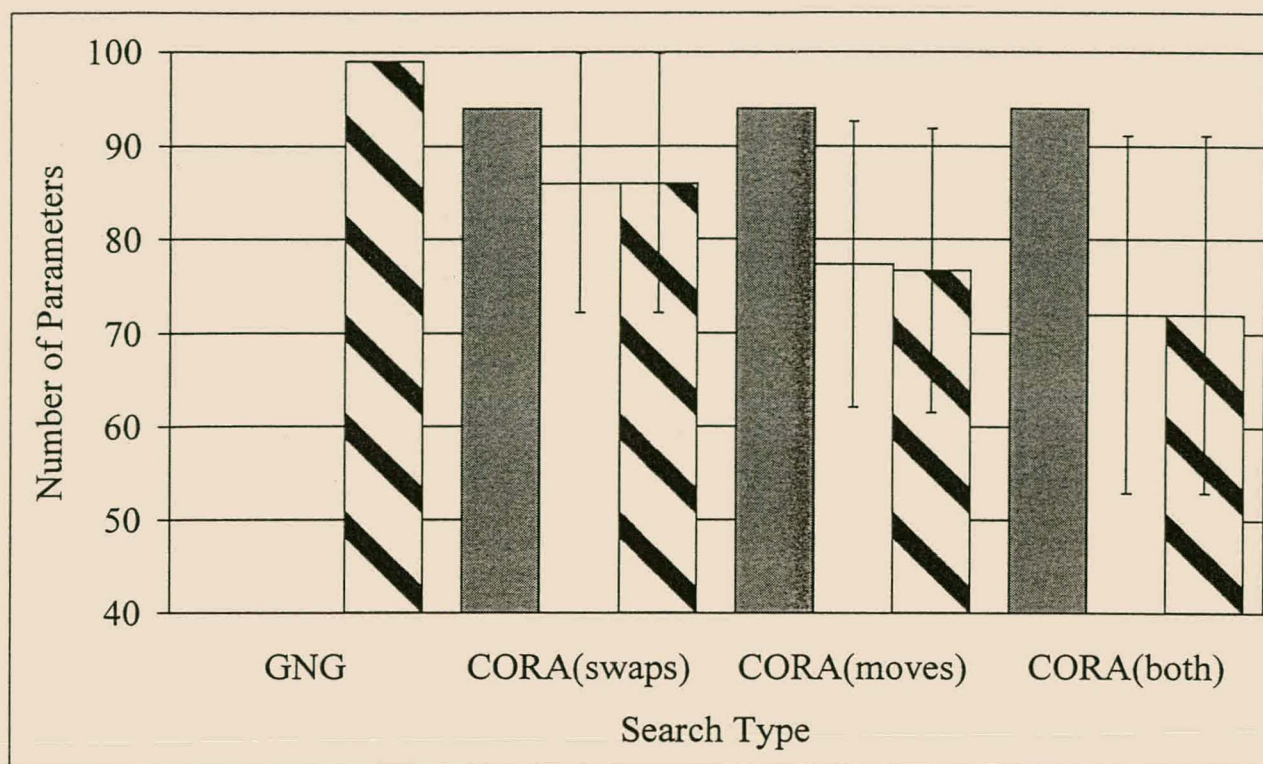
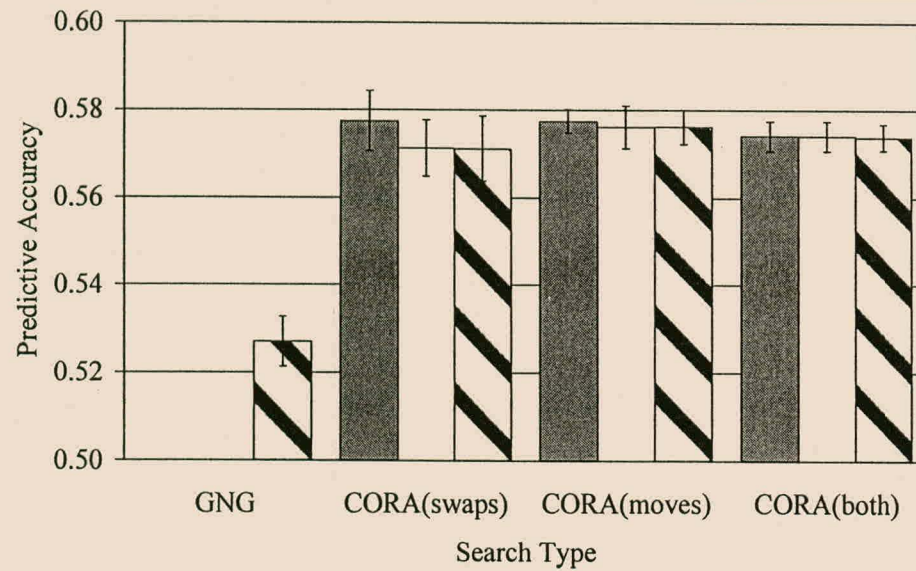
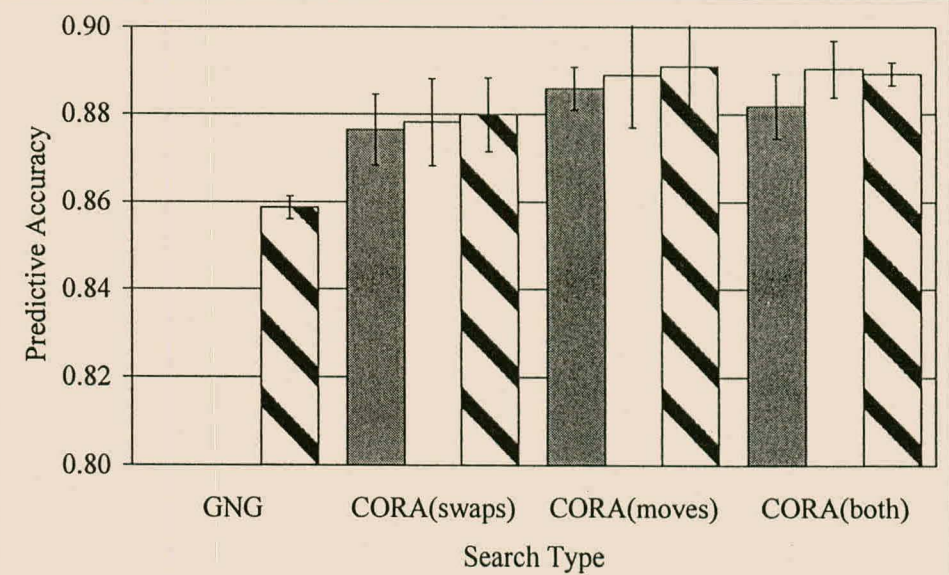


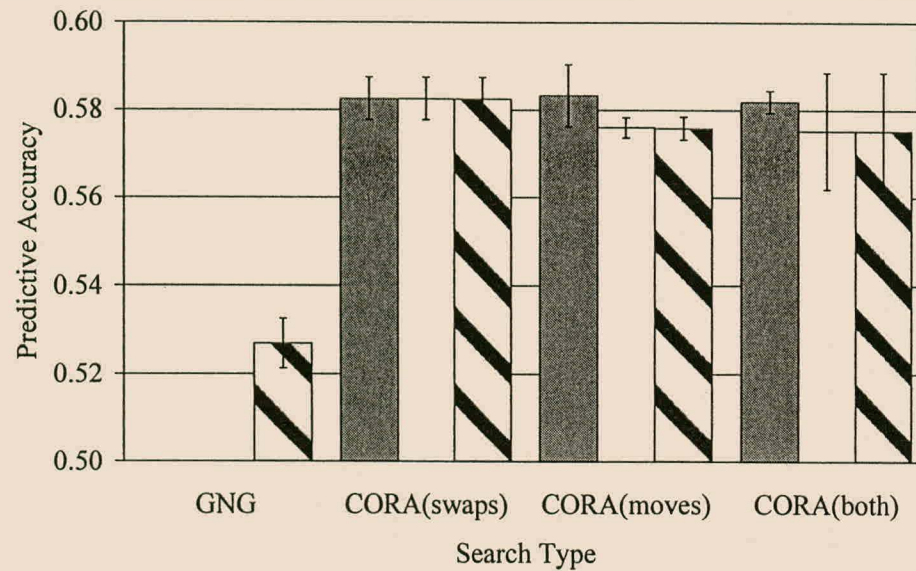
Figure B.53 No. of Parameters vs. SMB for Slugflow Data (6)



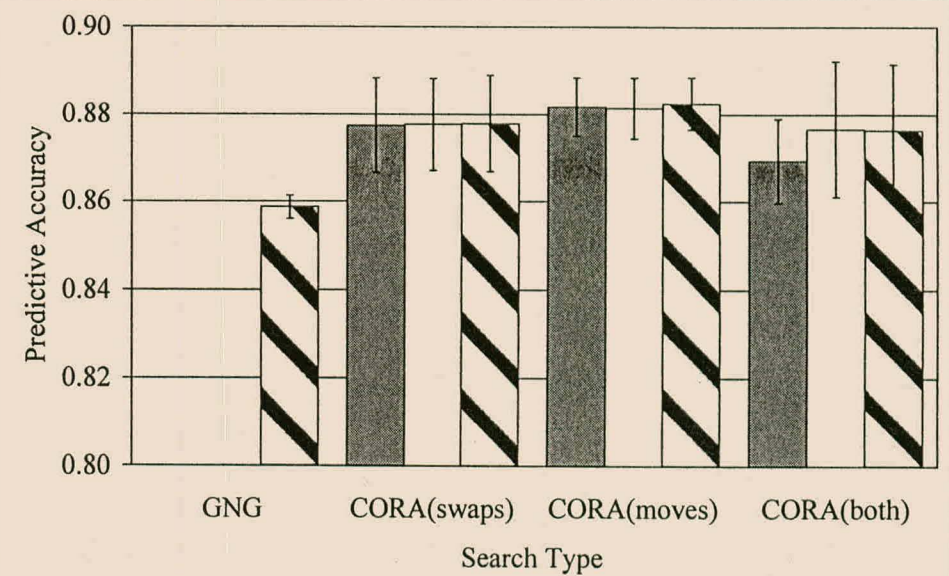
**Figure B.54 Accuracy vs. SMB for Abalone Problem (20)**



**Figure B.55 Accuracy vs. SMB for Auto Problem (22)**

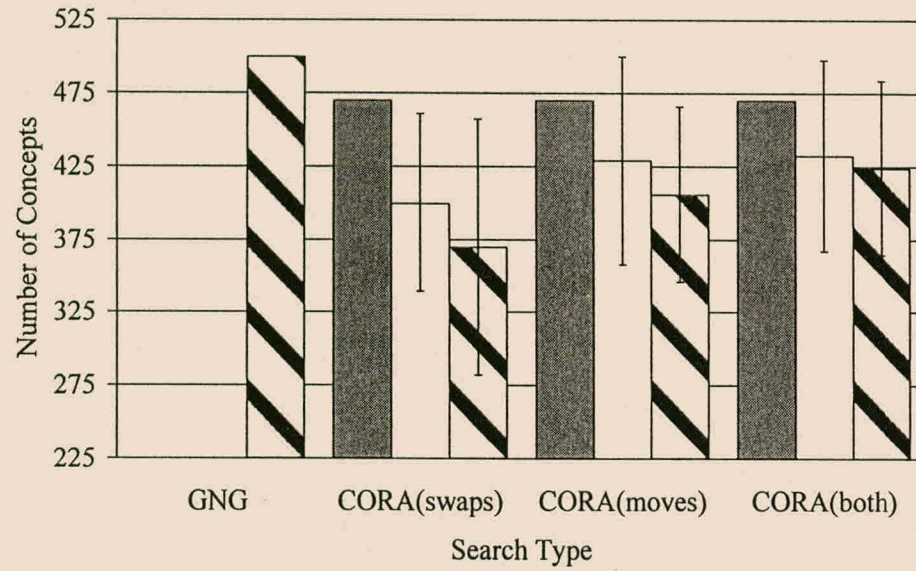


**Figure B.56 Accuracy vs. SMB for Abalone Problem (10)**

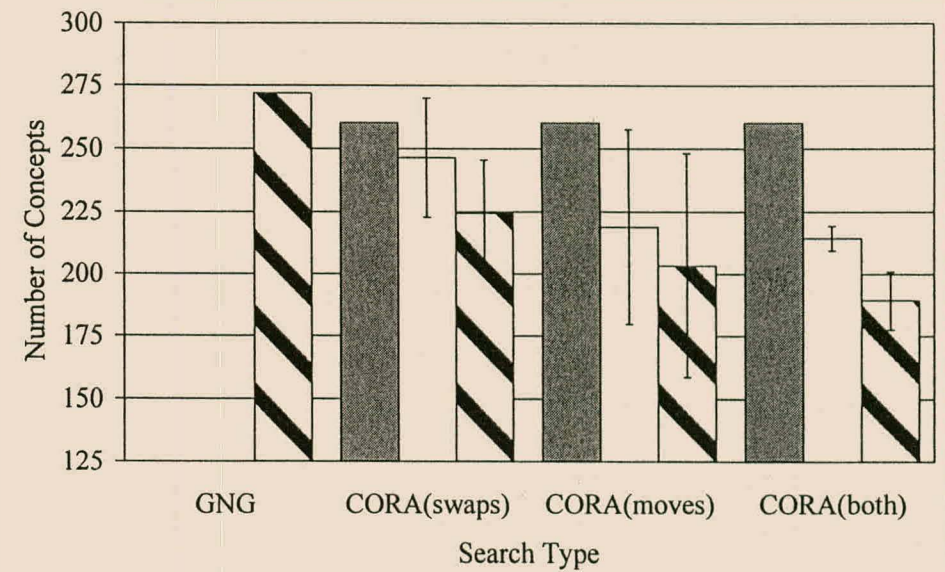


**Figure B.57 Accuracy vs. SMB for Auto Problem (17)**

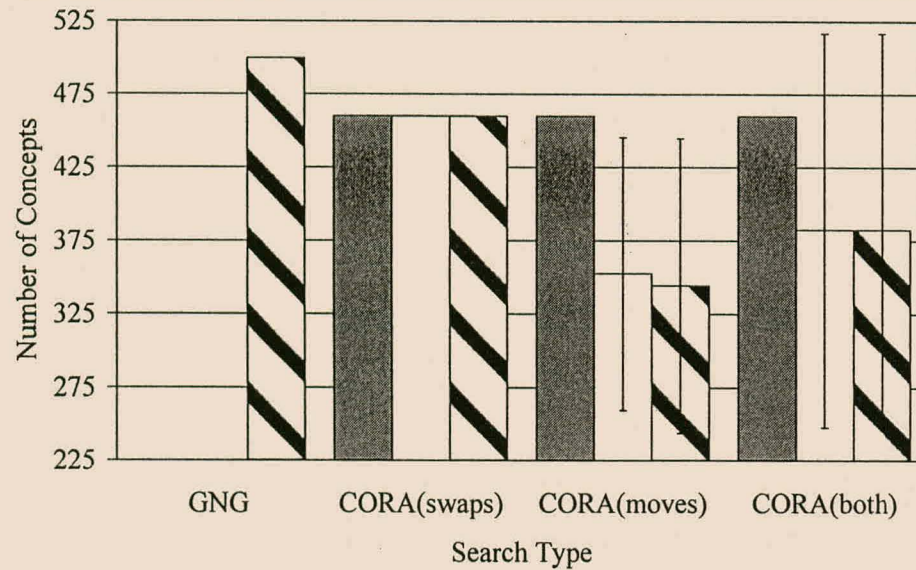




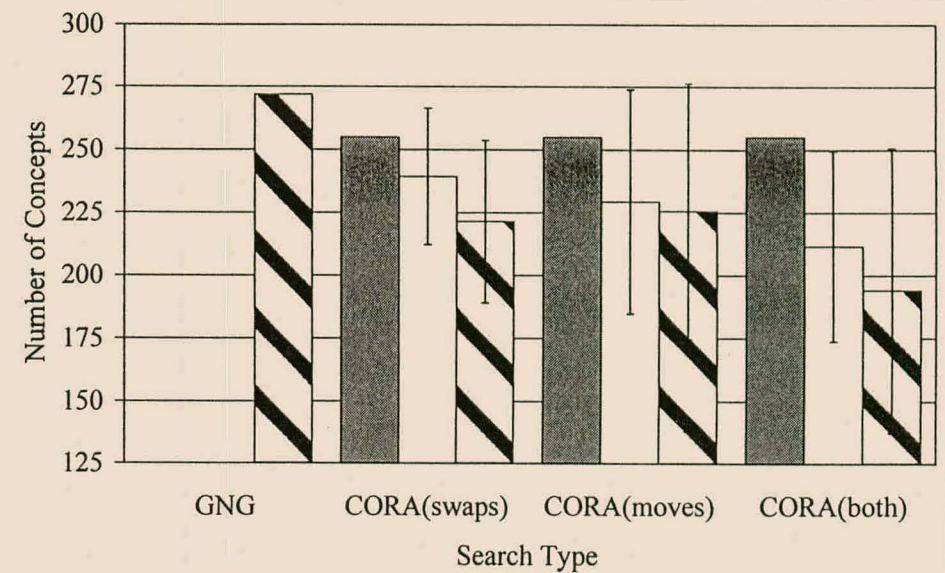
**Figure B.58 No. of Concepts vs. SMB for Abalone Problem (20)**



**Figure B.59 No. of Concepts vs. SMB for Auto Problem (22)**

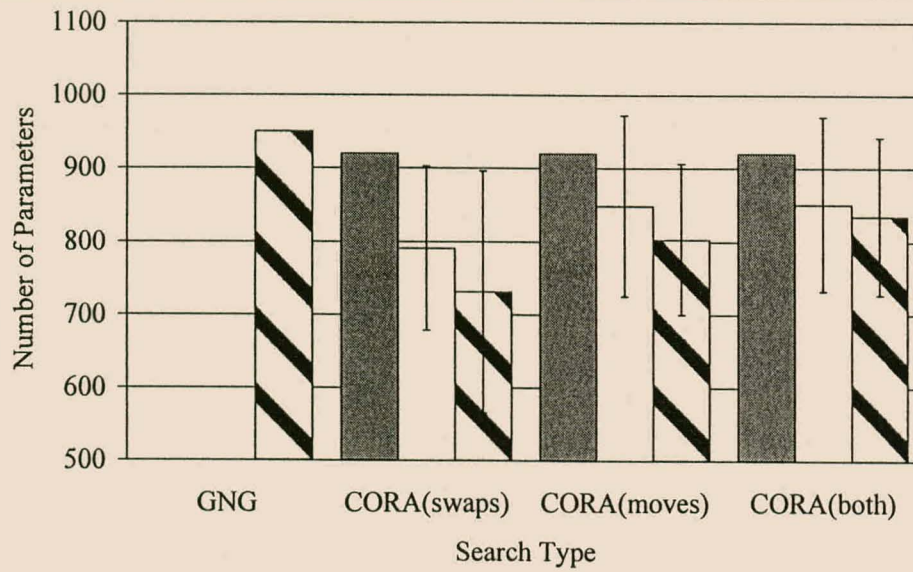


**Figure B.60 No. of Concepts vs. SMB for Abalone Problem (10)**

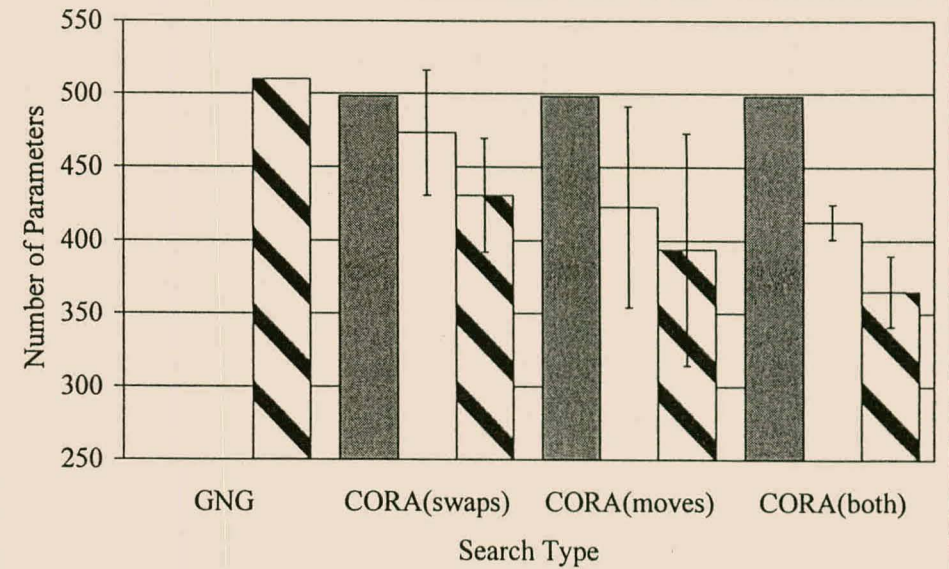


**Figure B.61 No. of Concepts vs. SMB for Auto Problem (17)**

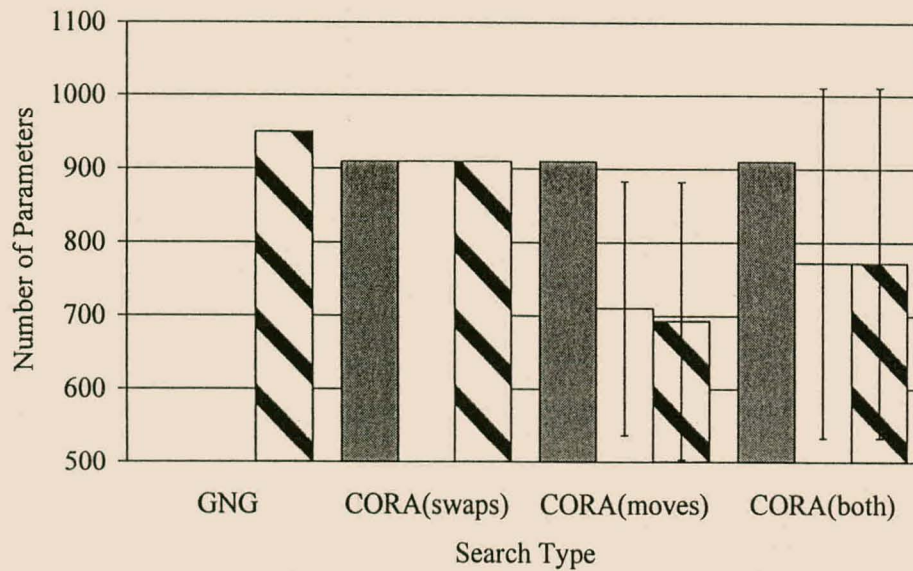




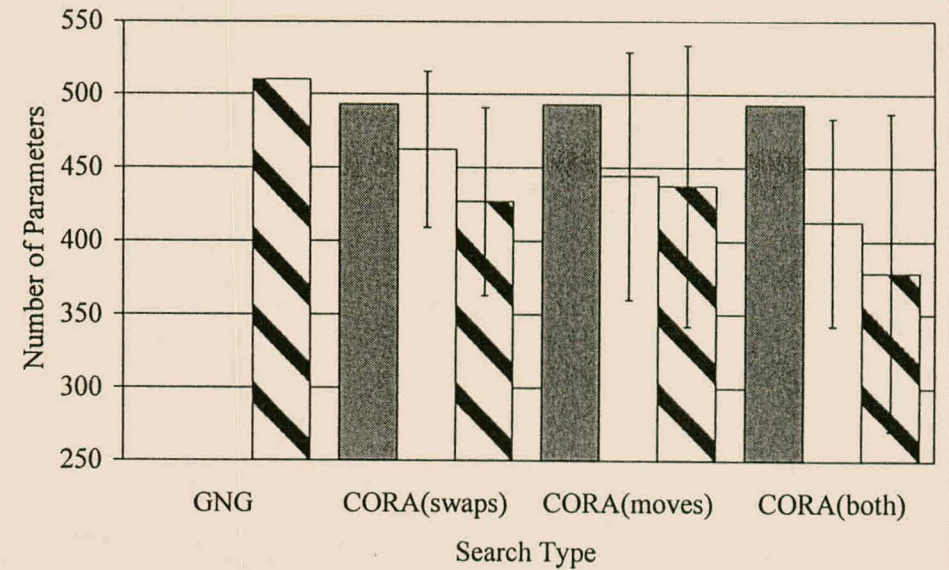
**Figure B.62 No. of Parameters vs. SMB for Abalone Problem (20)**



**Figure B.63 No. of Parameters vs. SMB for Auto Problem (22)**

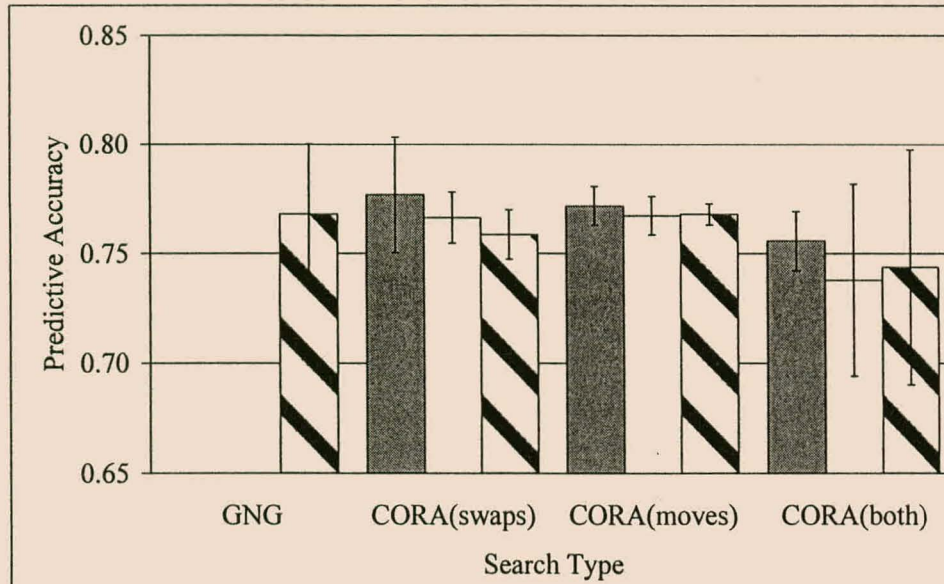


**Figure B.64 No. of Parameters vs. SMB for Abalone Problem (10)**

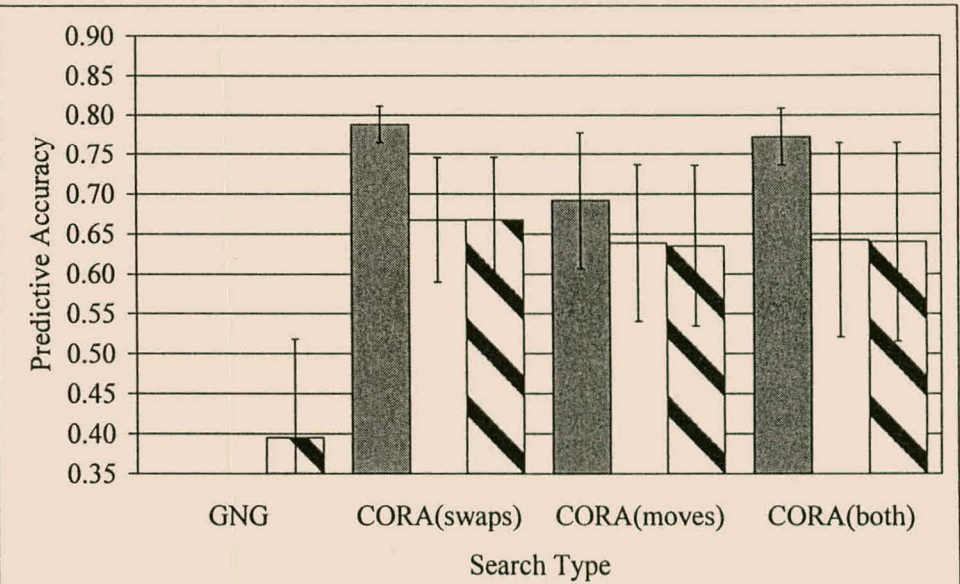


**Figure B.65 No. of Parameters vs. SMB for Auto Problem (17)**

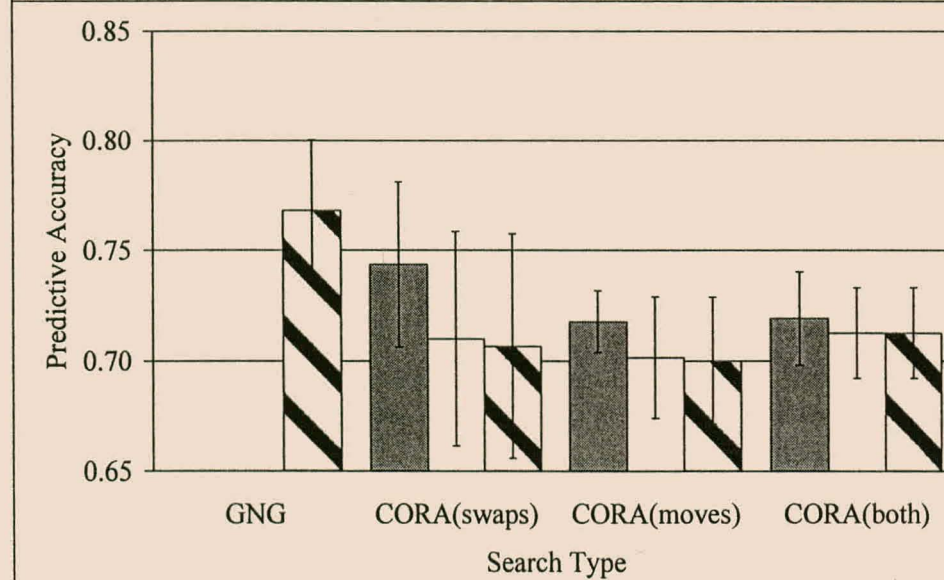




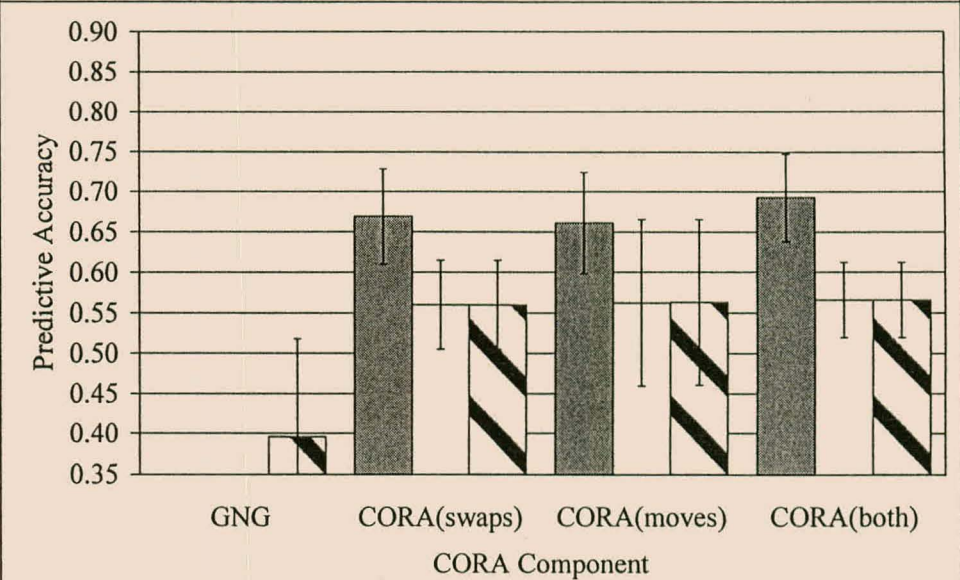
**Figure B.66 Accuracy vs. SMB for Housing Problem (20)**



**Figure B.67 Accuracy vs. SMB for Servo Problem (10)**

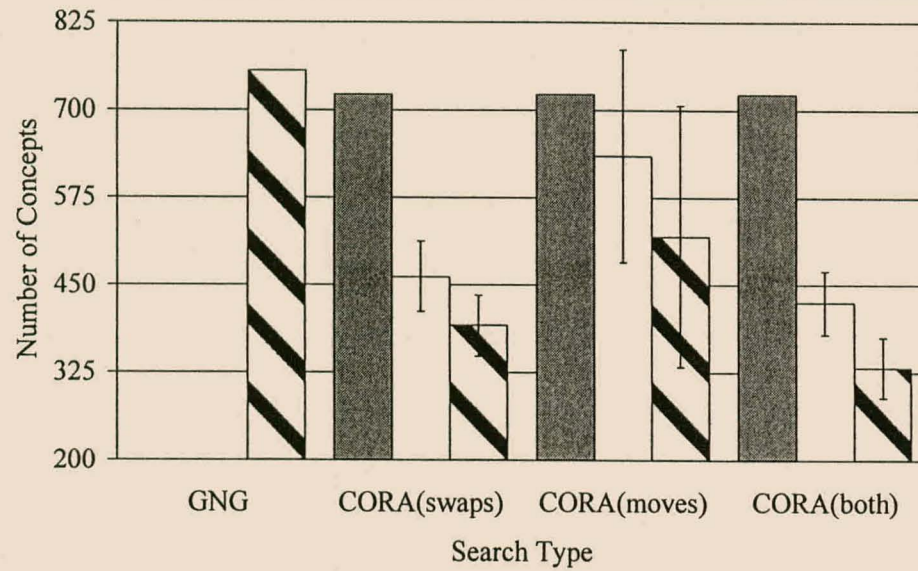


**Figure B.68 Accuracy vs. SMB for Housing Problem (10)**

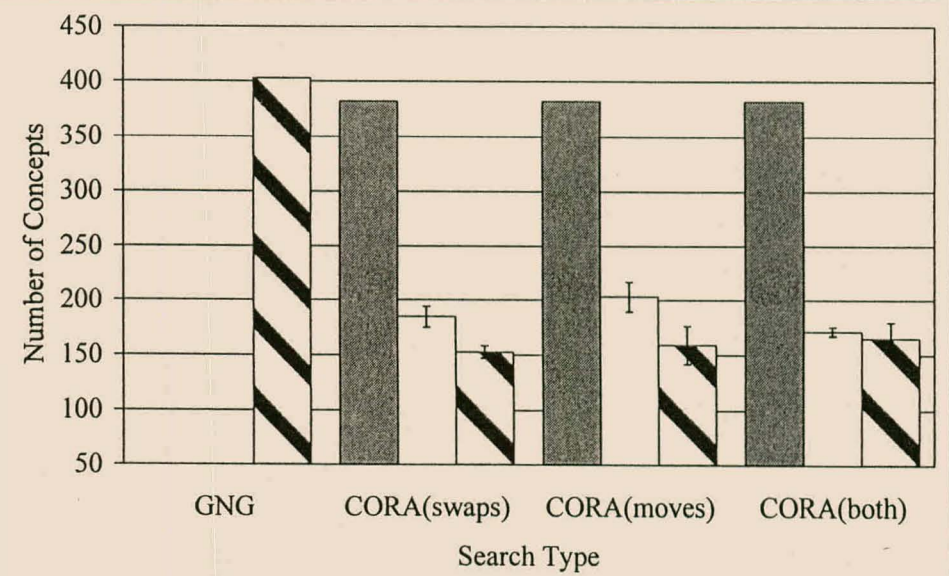


**Figure B.69 Accuracy vs. SMB for Servo Problem (5)**

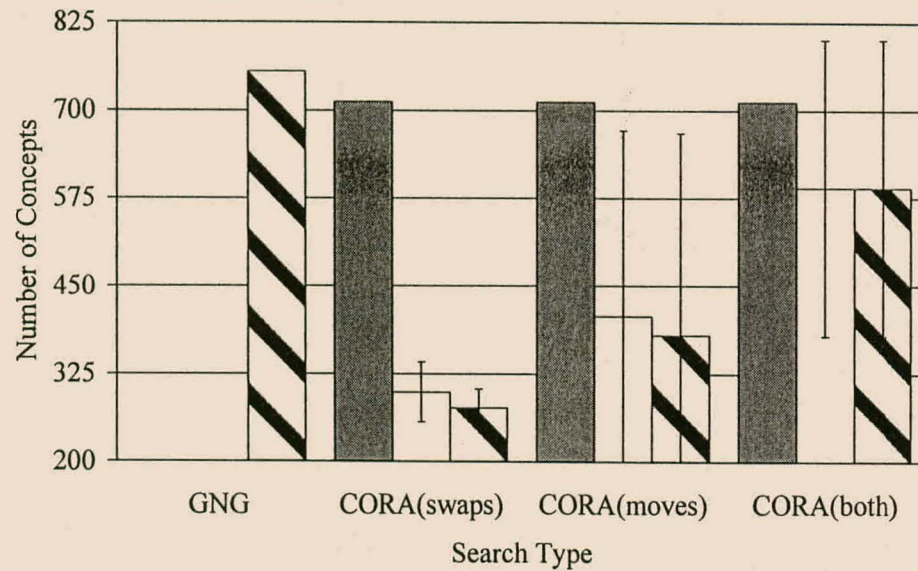




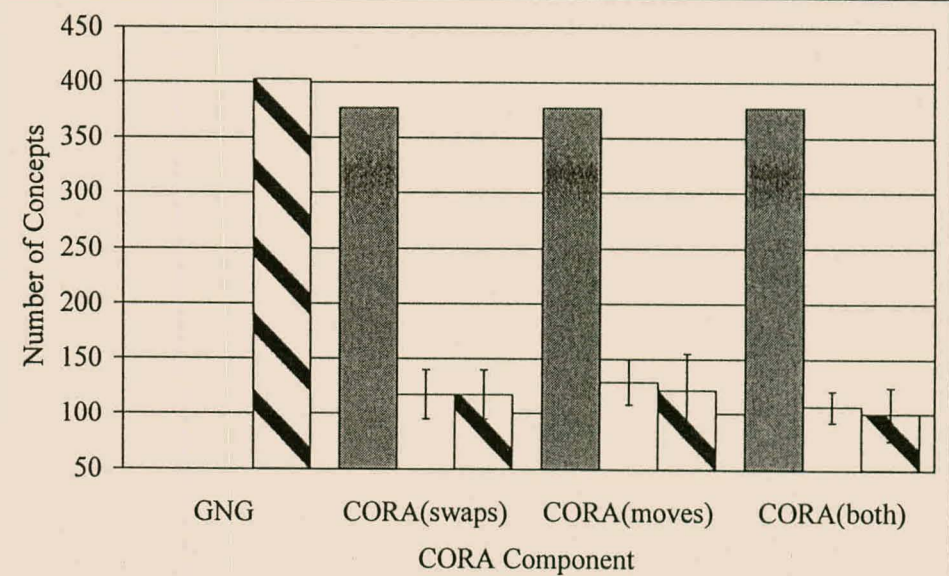
**Figure B.70 No. of Concepts vs. SMB for Housing Problem (20)**



**Figure B.71 No. of Concepts vs. SMB for Servo Problem (10)**

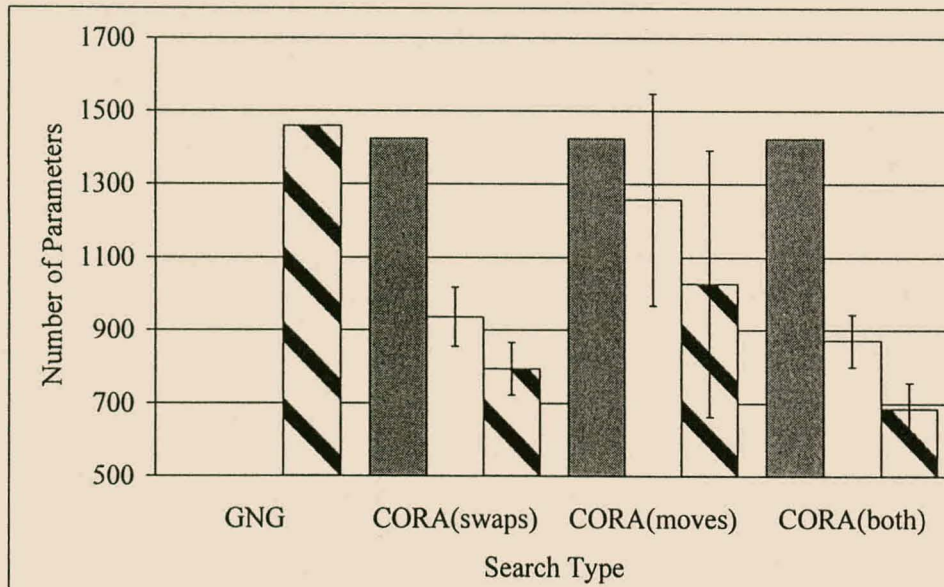


**Figure B.72 No. of Concepts vs. SMB for Housing Problem (10)**

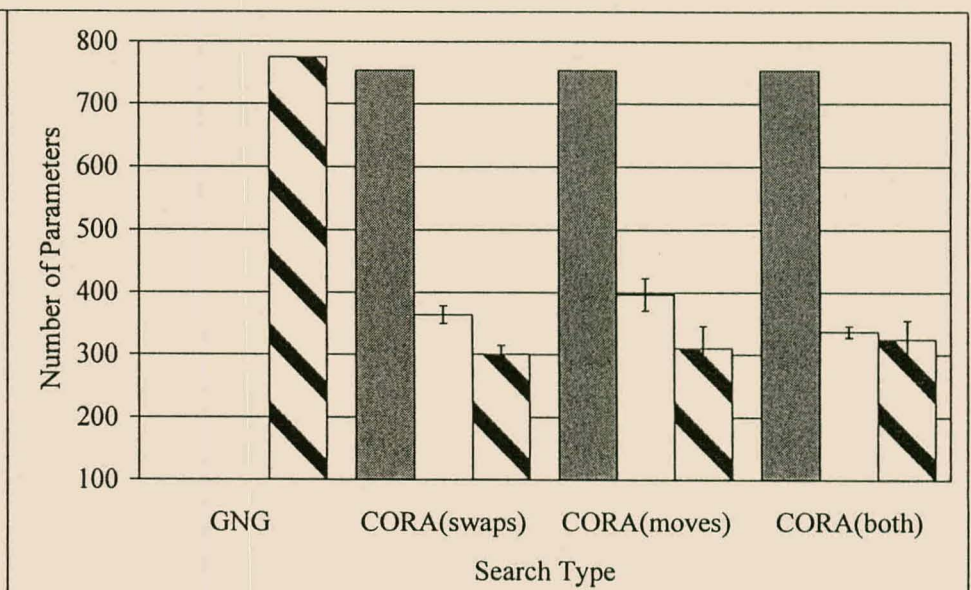


**Figure B.73 No. of Concepts vs. SMB for Servo Problem (5)**

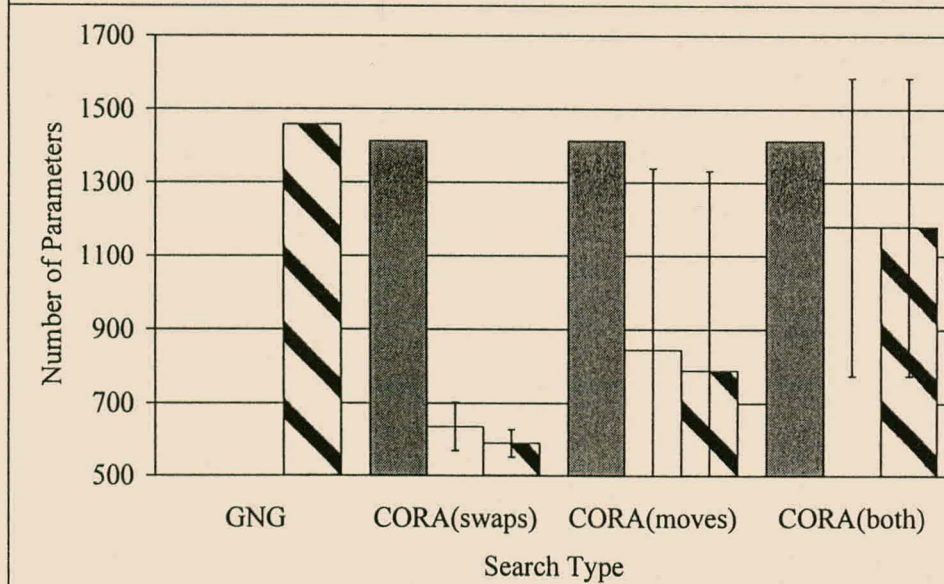




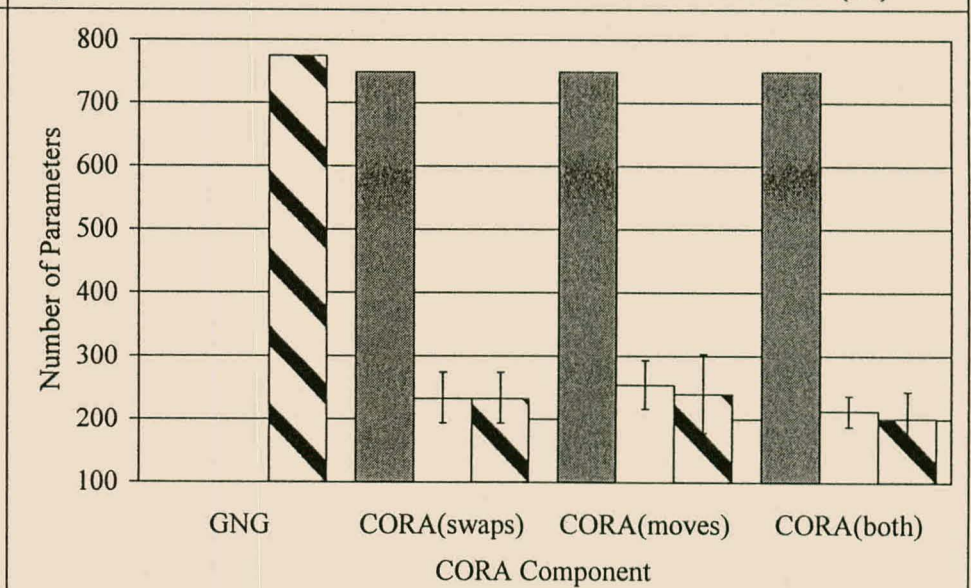
**Figure B.74 No. of Parameters vs. SMB for Housing Problem (20)**



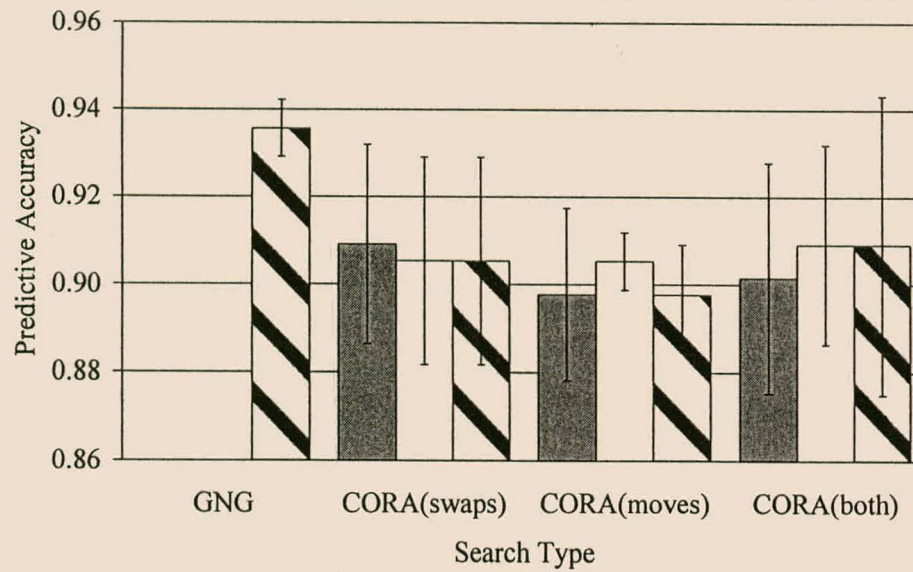
**Figure B.75 No. of Parameters vs. SMB for Servo Problem (10)**



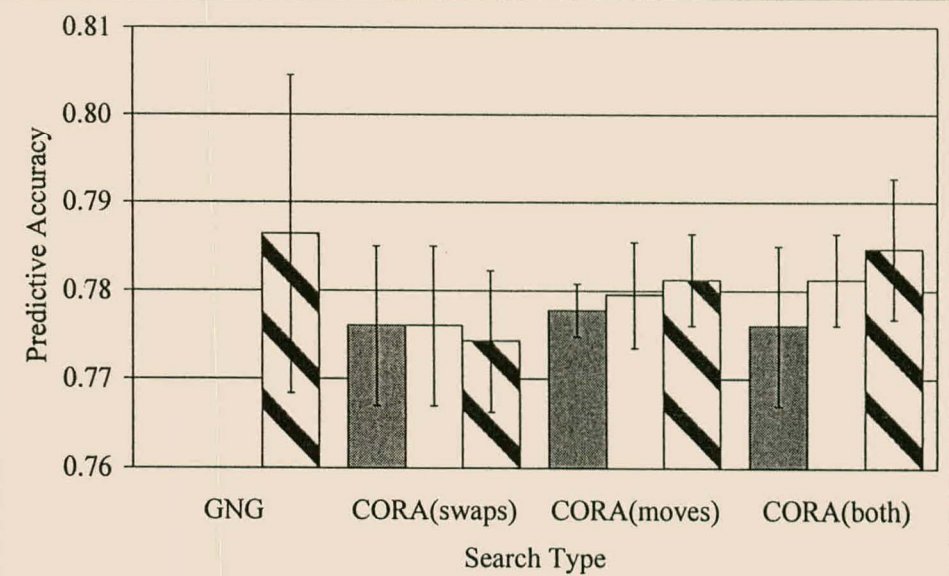
**Figure B.76 No. of Parameters vs. SMB for Housing Problem (10)**



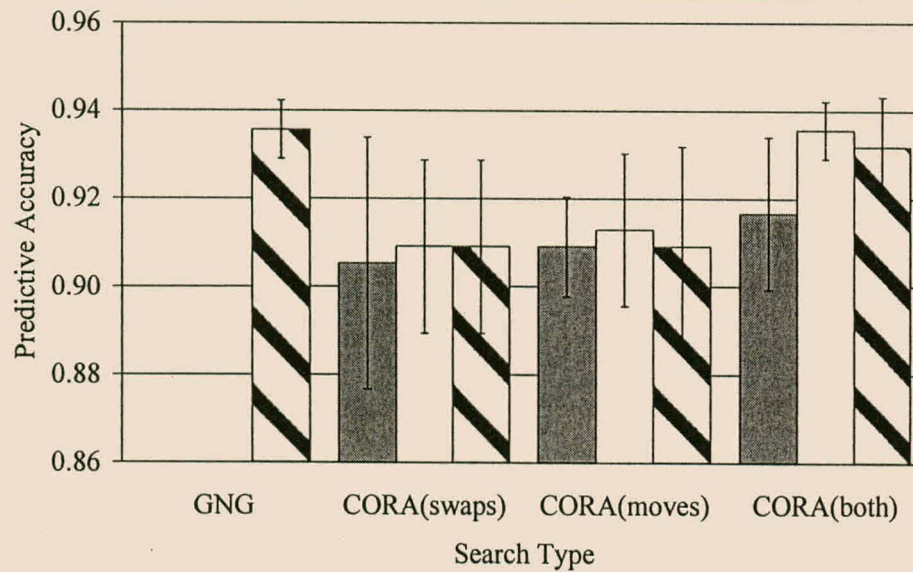
**Figure B.77 No. of Parameters vs. SMB for Servo Problem (5)**



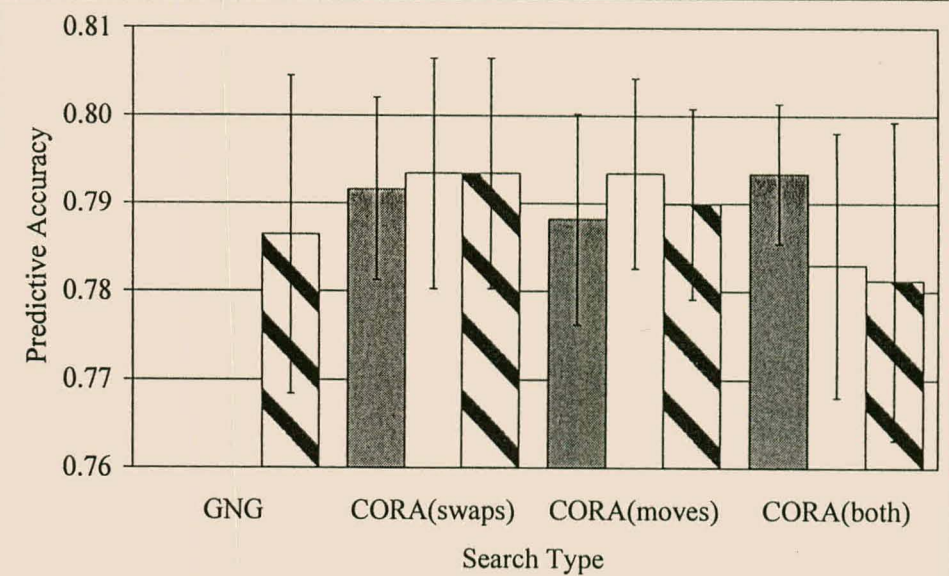
**Figure B.78 Accuracy vs. SMB for Ionosphere Problem (20)**



**Figure B.79 Accuracy vs. SMB for Pima Problem (10)**

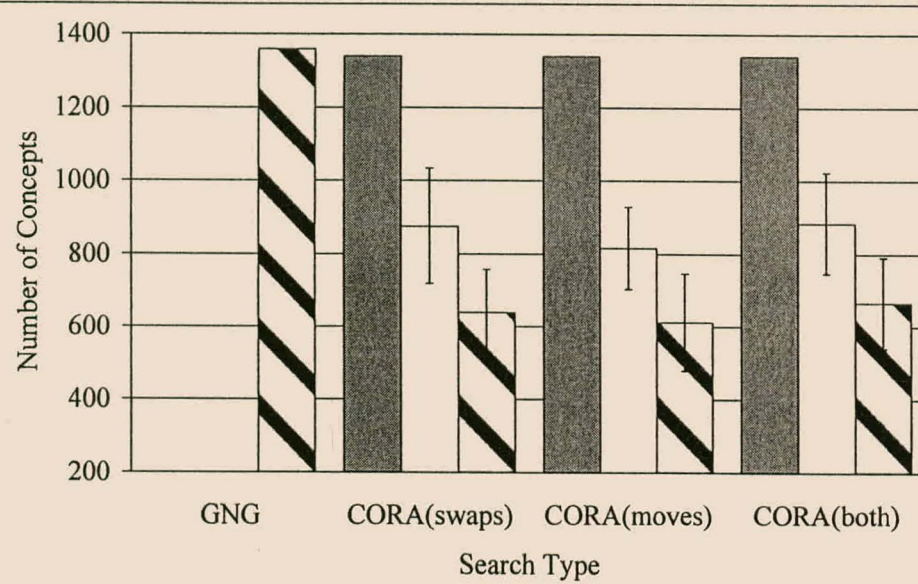


**Figure B.80 Accuracy vs. SMB for Ionosphere Problem (10)**

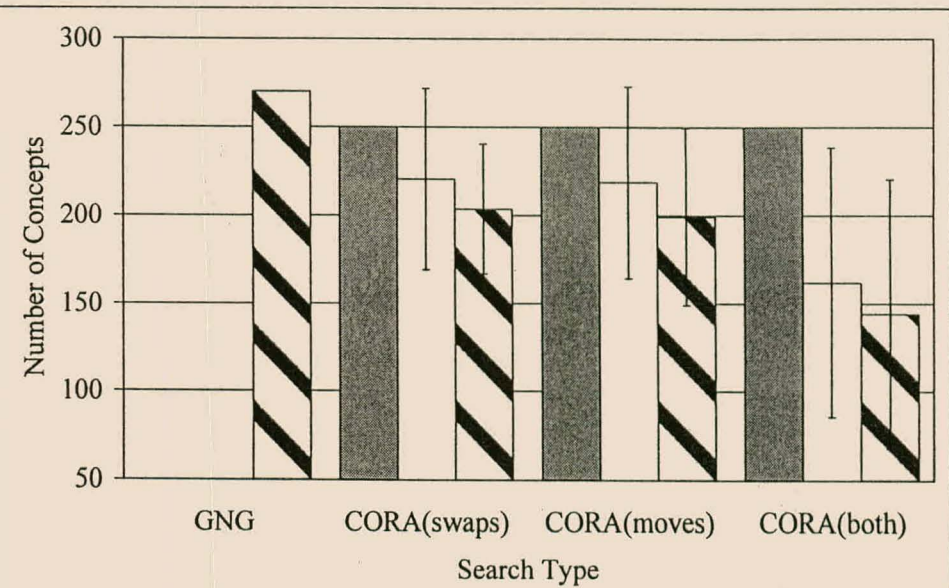


**Figure B.81 Accuracy vs. SMB for Pima Problem (5)**

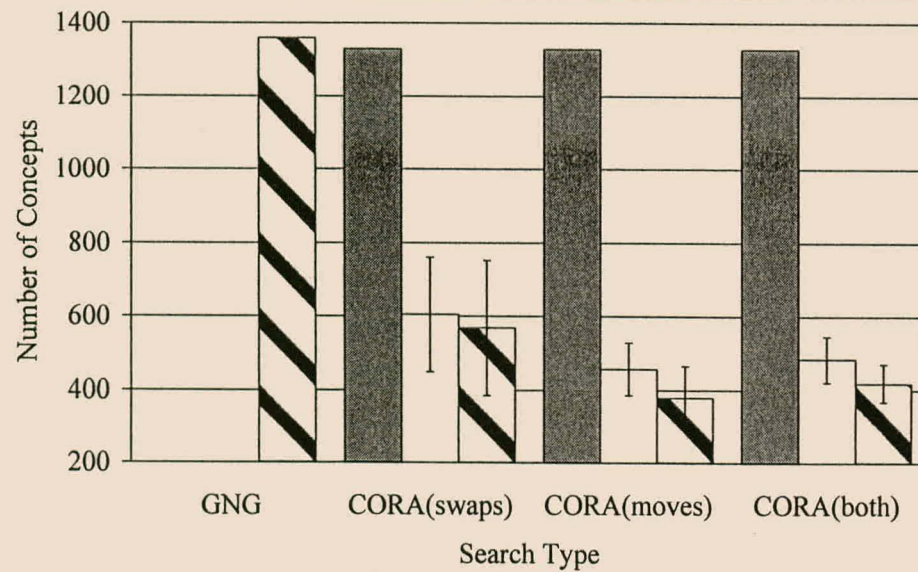




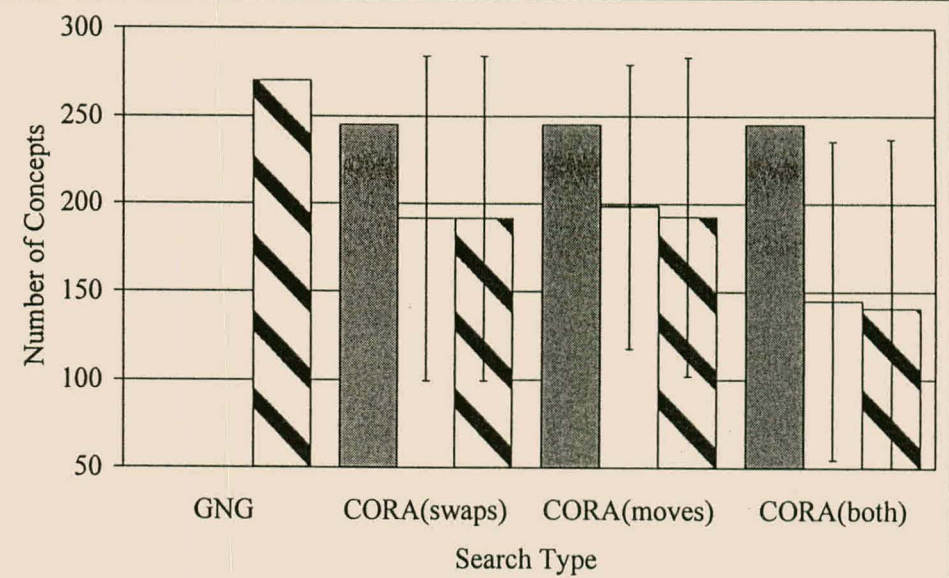
**Figure B.82 No. of Concepts vs. SMB for Ionosphere Data (20)**



**Figure B.83 No. of Concepts vs. SMB for Pima Problem (10)**



**Figure B.84 No. of Concepts vs. SMB for Ionosphere Data (10)**



**Figure B.85 No. of Concepts vs. SMB for Pima Problem (5)**



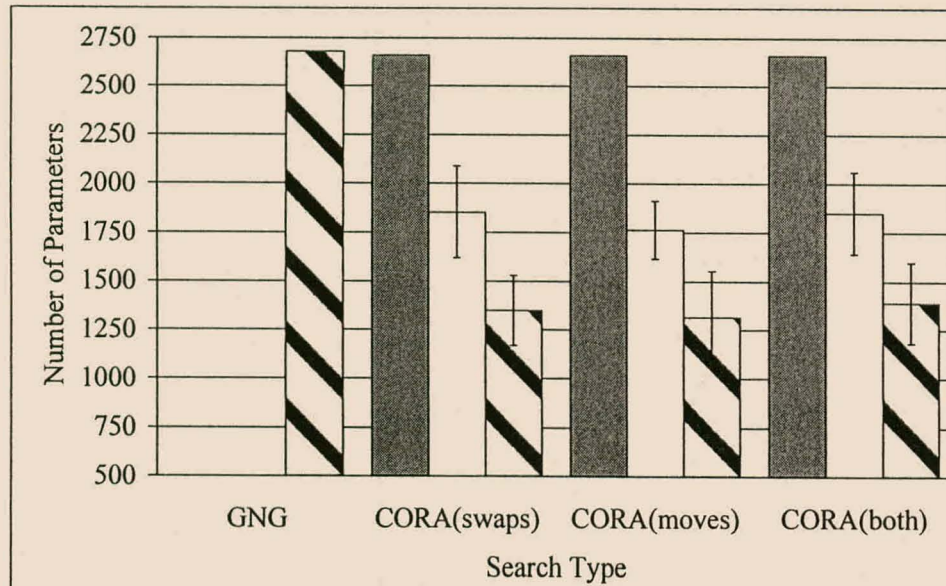


Figure B.86 No. of Parameters vs. SMB for Ionosphere Data (20)

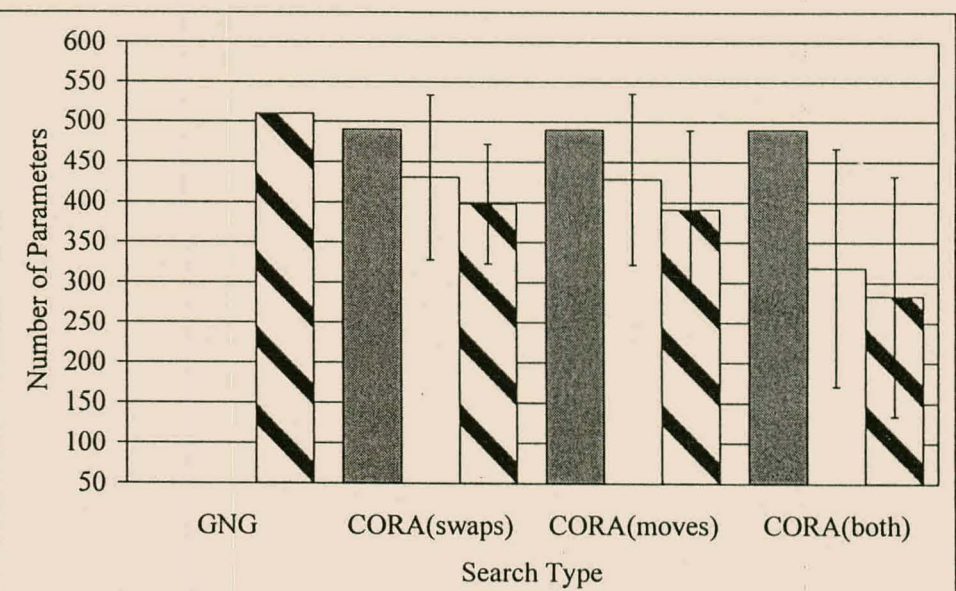


Figure B.87 No. of Parameters vs. SMB for Pima Problem (10)

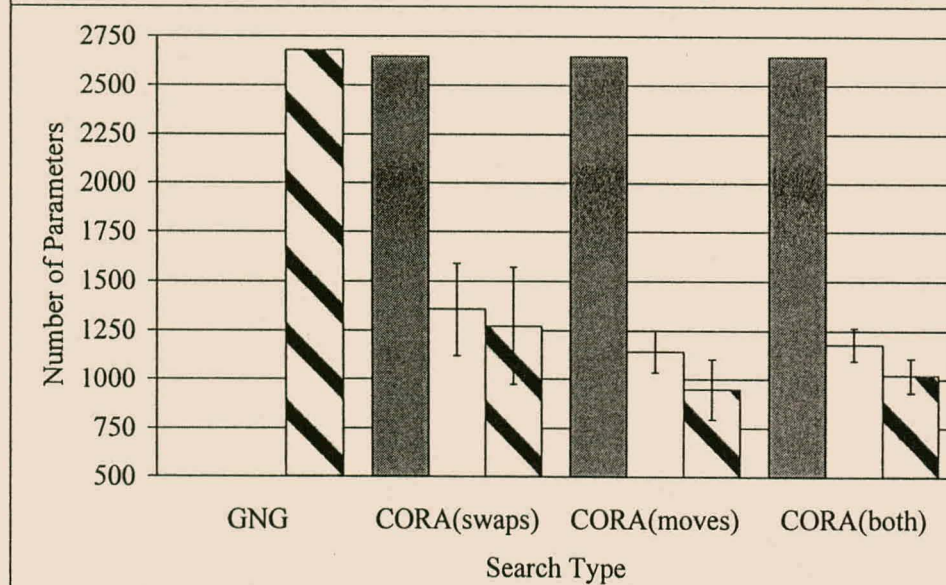


Figure B.88 No. of Parameters vs. SMB for Ionosphere Data (10)

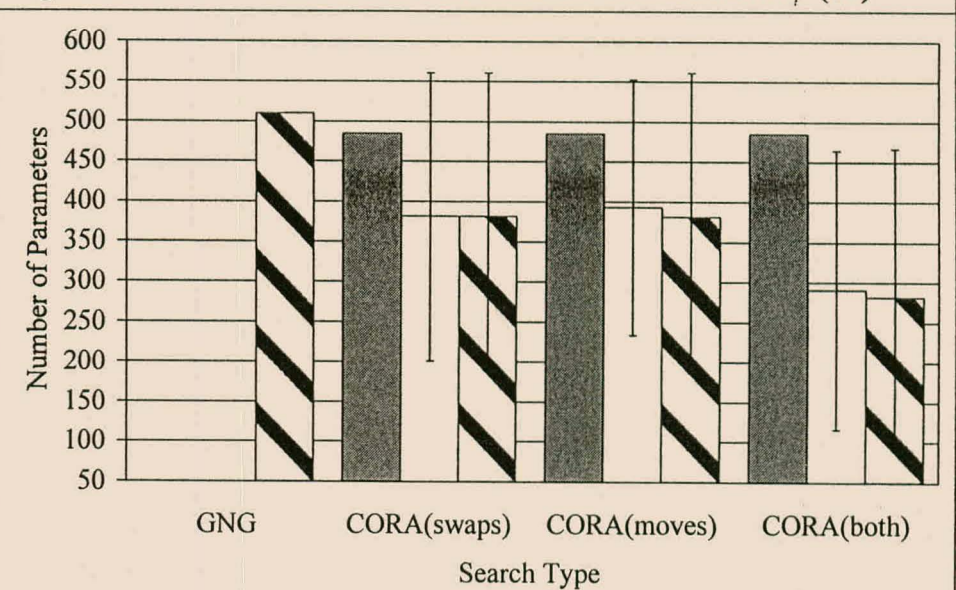
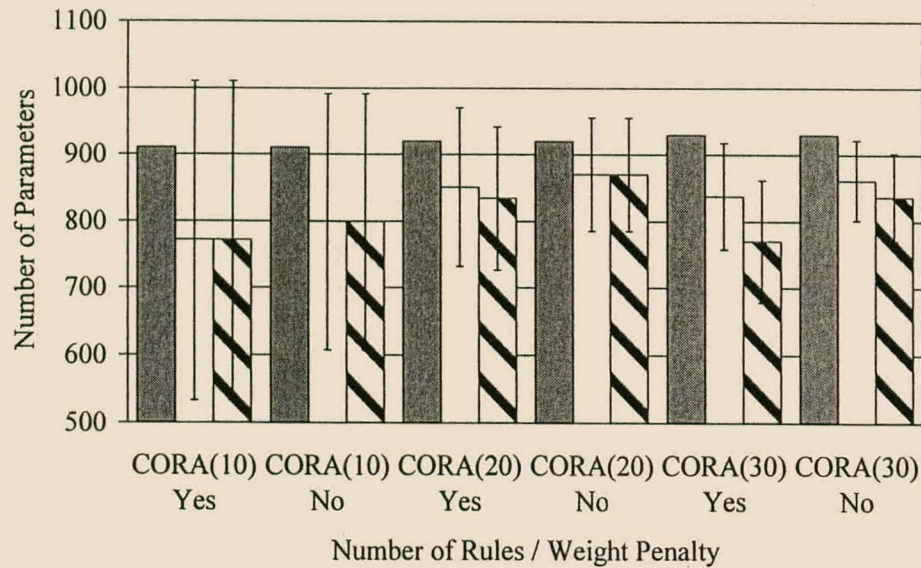
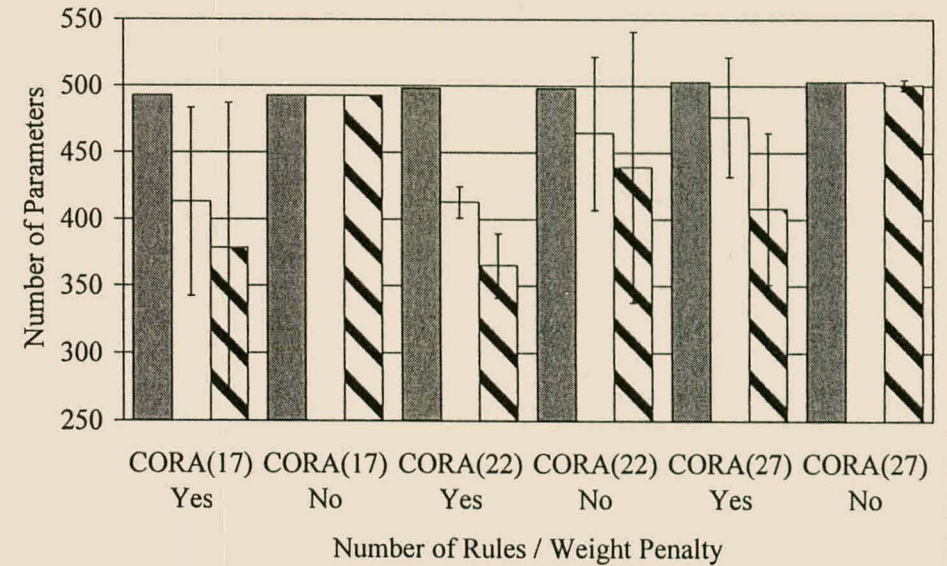


Figure B.89 No. of Parameters vs. SMB for Pima Problem (5)

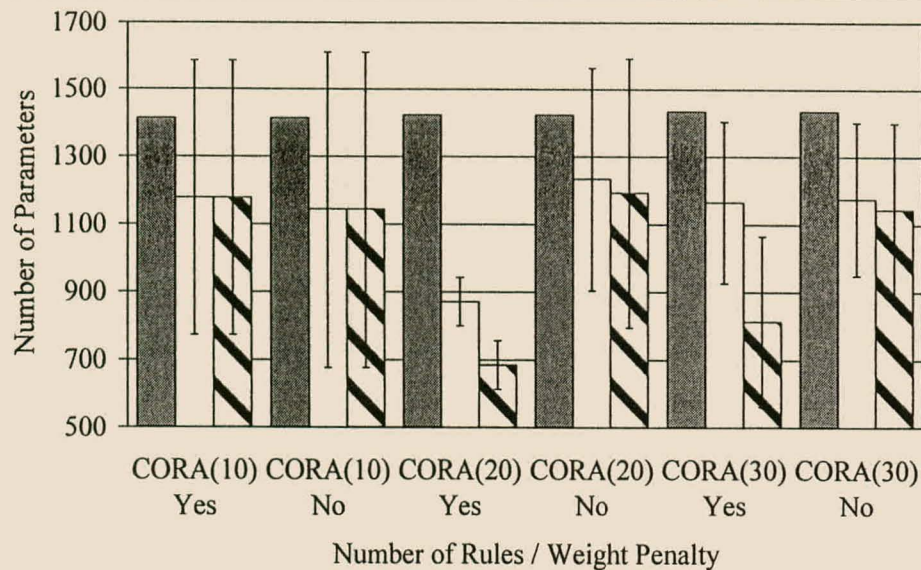




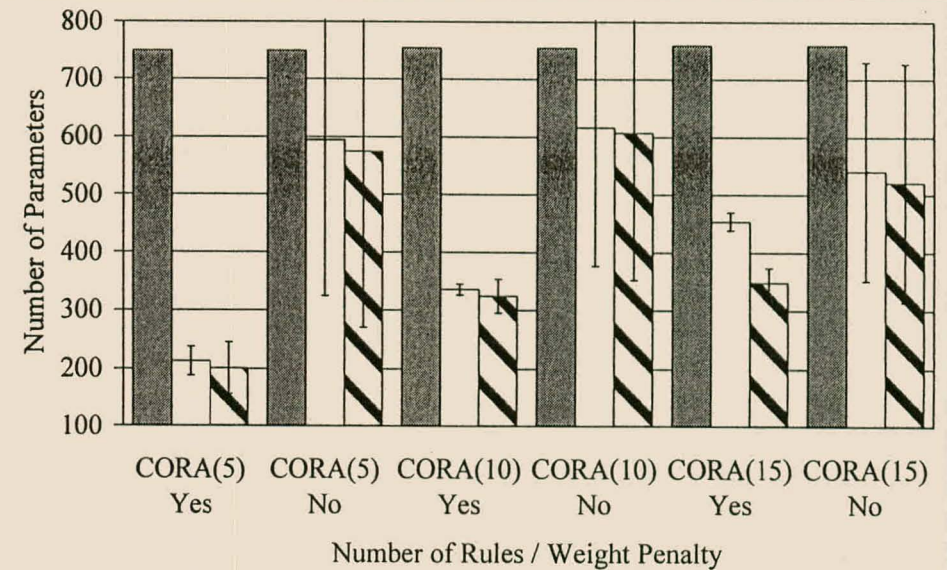
**Figure B.90 Parameters vs. Weight Penalty Use for Abalone Data**



**Figure B.91 Parameters vs. Weight Penalty Use for Auto Problem**

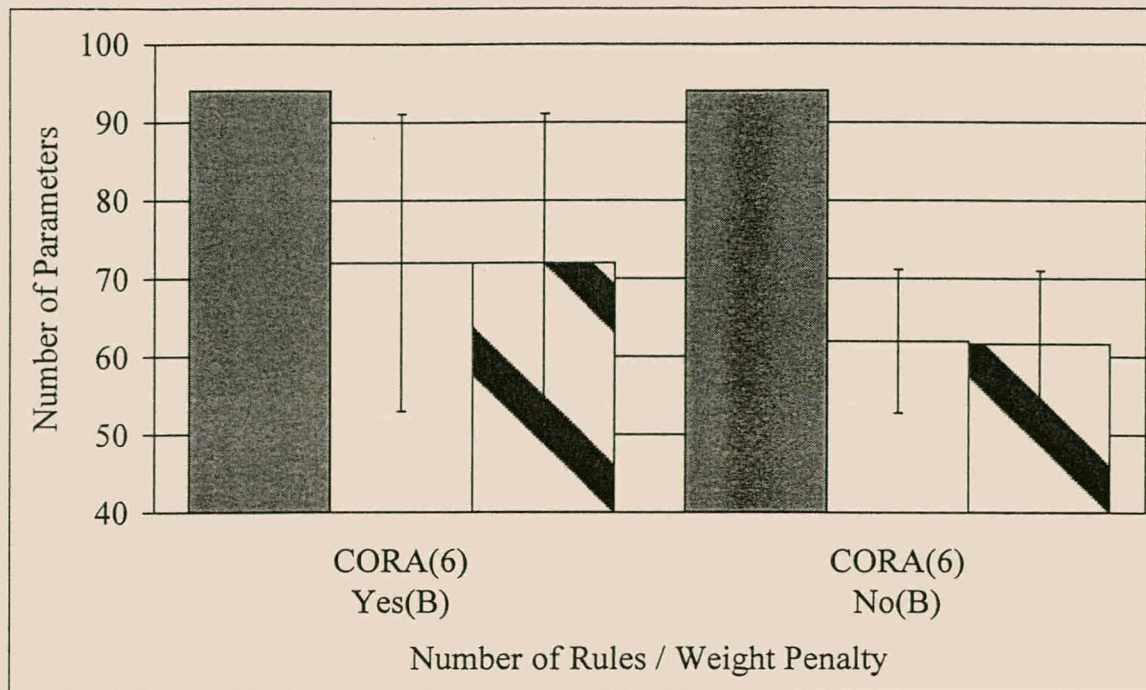


**Figure B.92 Parameters vs. Weight Penalty Use for Housing Data**

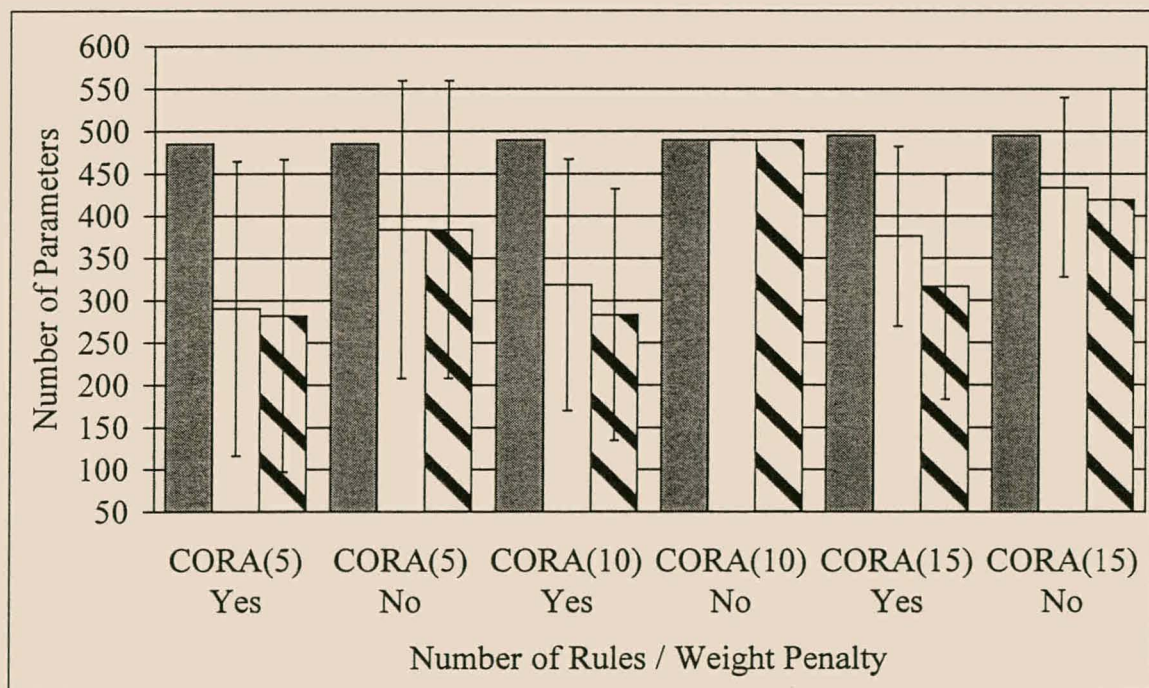


**Figure B.93 Parameters vs. Weight Penalty Use for Servo Problem**



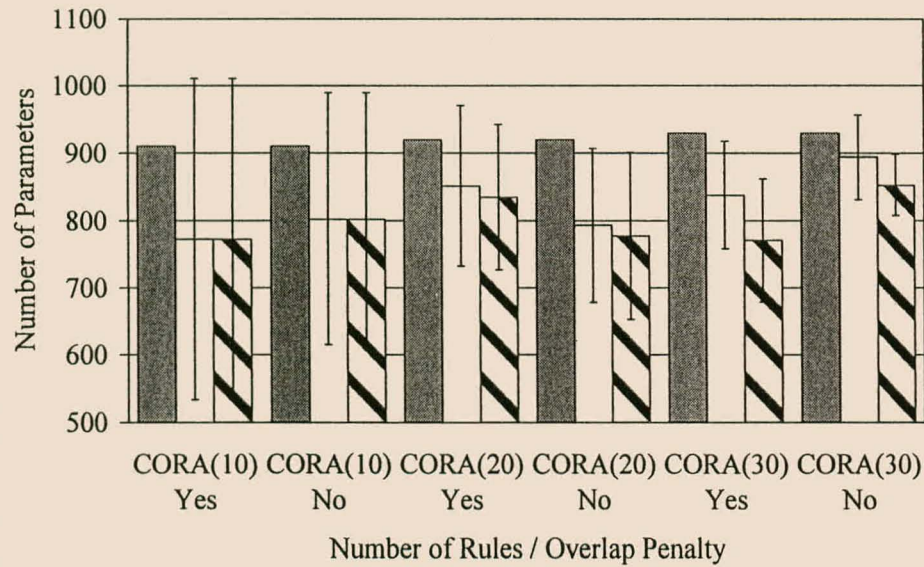


**Figure B.94 Parameters vs. Use of Weight Penalty for Slugflow Problem**

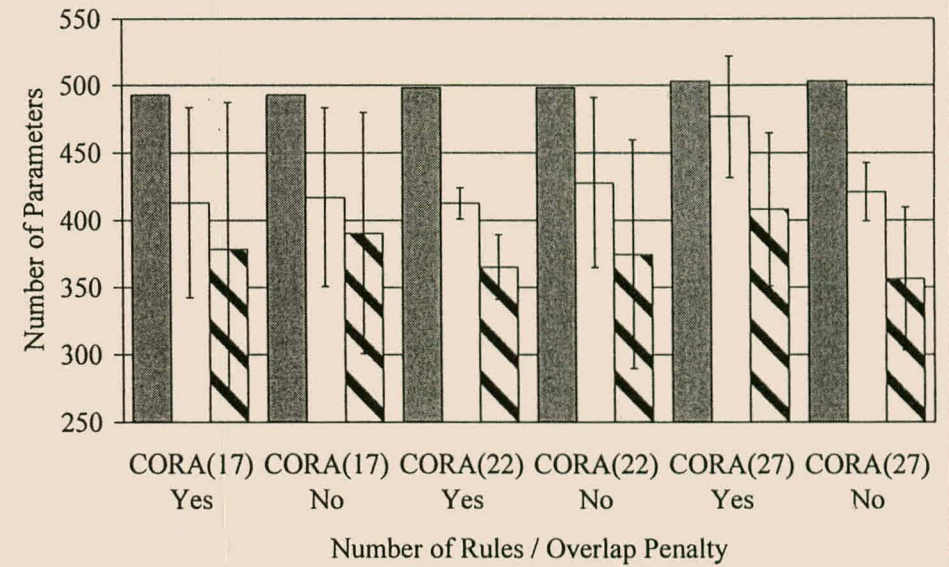


**Figure B.95 Parameters vs. Use of Weight Penalty for Pima Problem**

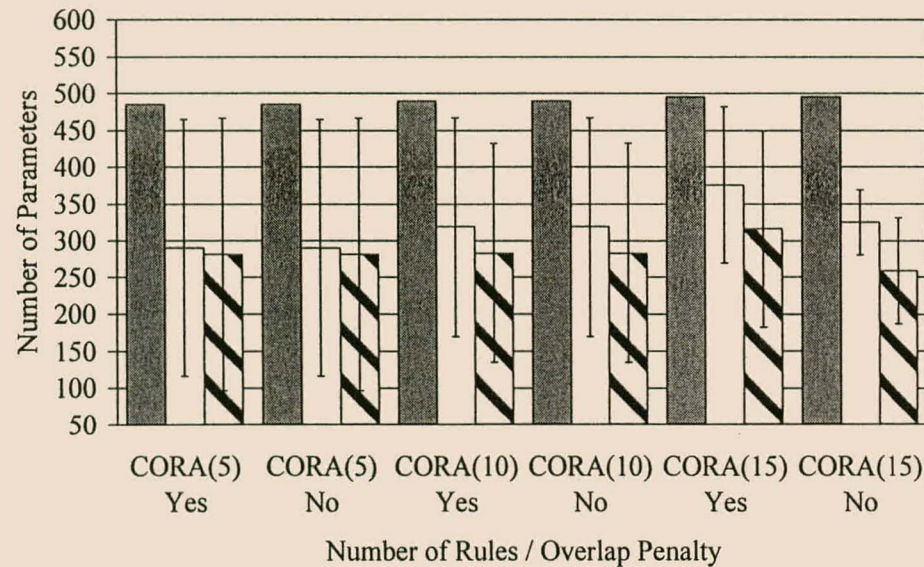




**Figure B.96 Parameters vs. Use of Overlap Penalty (Abalone Data)**



**Figure B.97 Parameters vs. Use of Overlap Penalty (Auto Data)**



**Figure B.98 Parameters vs. Use of Overlap Penalty (Pima Data)**

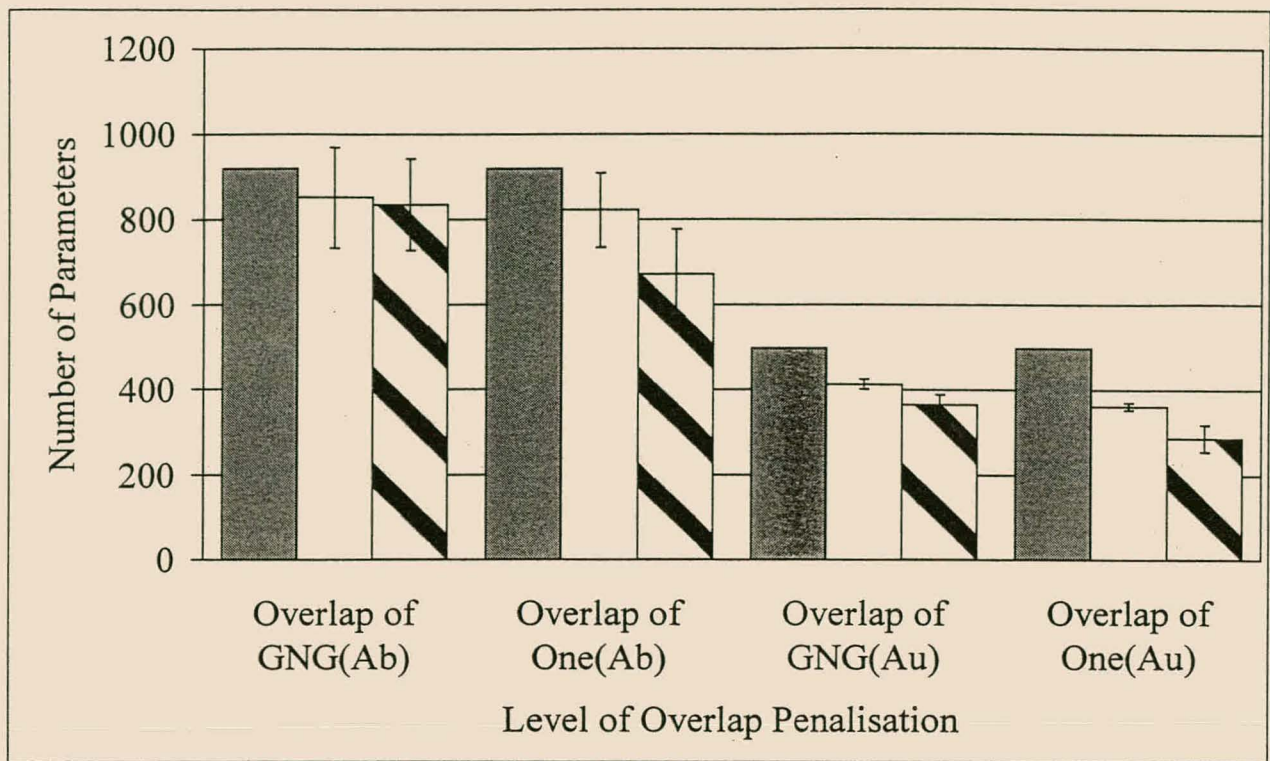


Figure B.99 Parameters vs. Overlap Level (Abalone and Auto Problems)

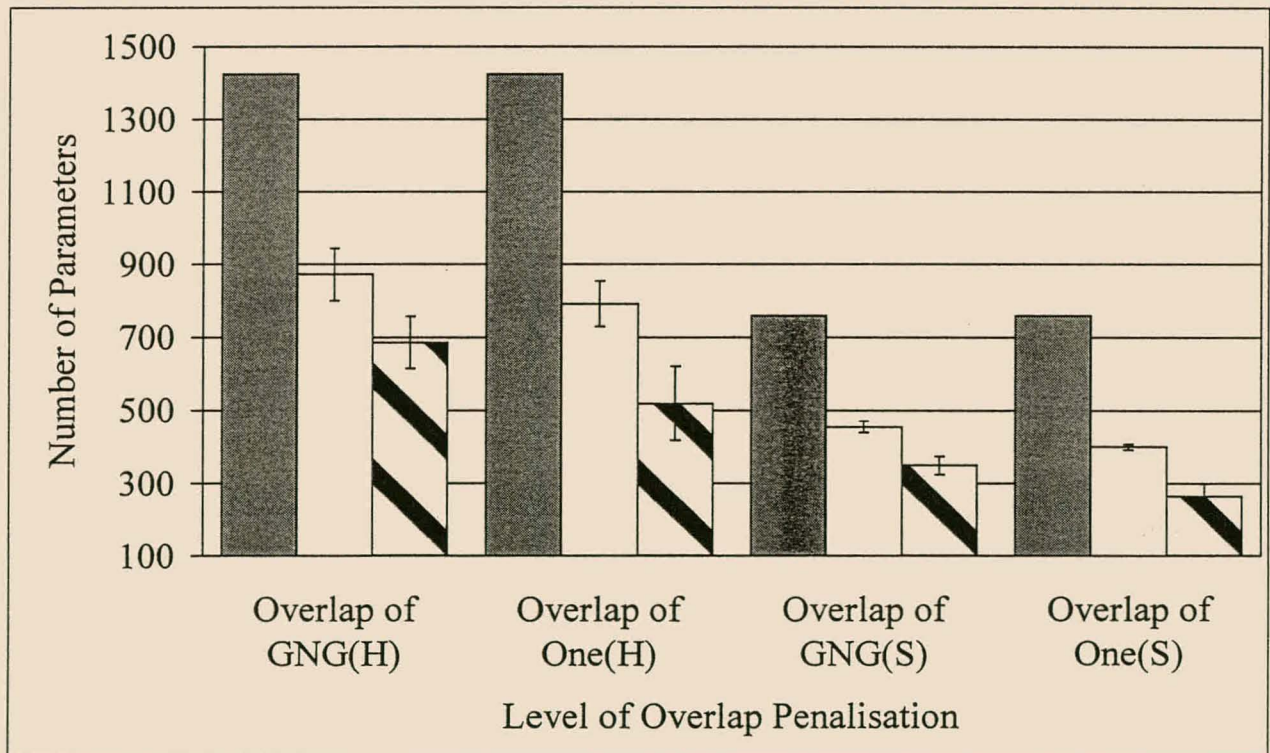


Figure B.100 Parameters vs. Overlap Level (Housing and Servo Problems)

# Appendix C

## Algorithmic Training Parameters Used in the Evaluation of the Autocatalytic Data

This appendix lists (where applicable) the training parameter values that were used by the modelling techniques employed in the evaluation of the Autocatalytic data in chapter 7. The techniques and models that were considered in chapter 7 are the CART regression tree algorithm, the Growing Neural Gas and  $k$ -means radial basis function network training algorithms, multivariate linear regression, a multilayer perceptron trained using the generalised delta rule, the MARS algorithm and finally the CORA algorithm.

As described in section 7.3.1, three training runs were performed with each stochastic modelling technique for the final testing experiments. The purpose of performing three runs was to determine the variation of results if an algorithm uses different random seeds.

### C.1 The CART Algorithm

The training parameter settings used by the CART algorithm in chapter 7 are as follows. Least squares error minimisation was used to find the best attribute and attribute value upon which to split at a decision node. During tree induction, all surrogates counted equally and a maximum of five surrogates were used.



## C.2 The Growing Neural Gas Algorithm

In most cases the default training parameter settings given for the Growing Neural Gas algorithm in appendix A, specifically table A.2 in section A.3, were used in the experiments performed for chapter 7. The exceptions are the number of training epochs and the maximum number of cell nodes that could be added to the radial basis function network's hidden layer. For the three different random seed runs the number of training epochs was respectively 60, 60 and 85. The maximum number of cell nodes was set at 30 for all runs.

## C.3 The CORA Algorithm

As described in section 7.3.2, two variants of the CORA algorithm were evaluated. The first was the standard CORA algorithm. The second variant attempts to minimise attribute space overlap in an identical fashion to the algorithm described in section 6.3.1. The training parameter values for the two CORA variants are summarised in table C.1. Both CORA algorithmic variants used the same training parameters. Where parameter values are not specified the same values as those given in table A.3 in appendix A were used.

Setting or Parameter	Value
Number of Reactive Tabu Search Iterations	2000
Initial Number of Fuzzy Rules to Assemble	20
Consequent Magnitude Penalisation Factor	0.03
Attribute Space Overlap Penalisation Factor	0.5
Minimum Swap / Move Threshold	0.9

**Table C.1 CORA Algorithm Training Parameter Values Used in Chapter 7**

## C.4 The $k$ -means Radial Basis Function Network

The training parameters that were used for the three random seed runs performed by the  $k$ -means radial basis function network algorithm are as follows. For all three random seed runs the number of hidden layer radial basis functions was 38 nodes. Furthermore, 152 training iterations and a width factor of two were used for all three runs.



## C.5 The Multilayer Perceptron

All experiments were performed using a multilayer perceptron that contained a single hidden layer. The hidden layer contained five nodes that were each described by a tanh transfer function. In all experiments each multilayer perceptron was trained for a total of 120000 epochs.

## C.6 Multivariate Adaptive Regression Splines

As mentioned in section 7.3.2, the models generated by two variants of the MARS algorithm were evaluated. One uses bootstrapping and the other does not. Apart from this factor the same training parameter values specified in table A.6 in appendix A were employed.